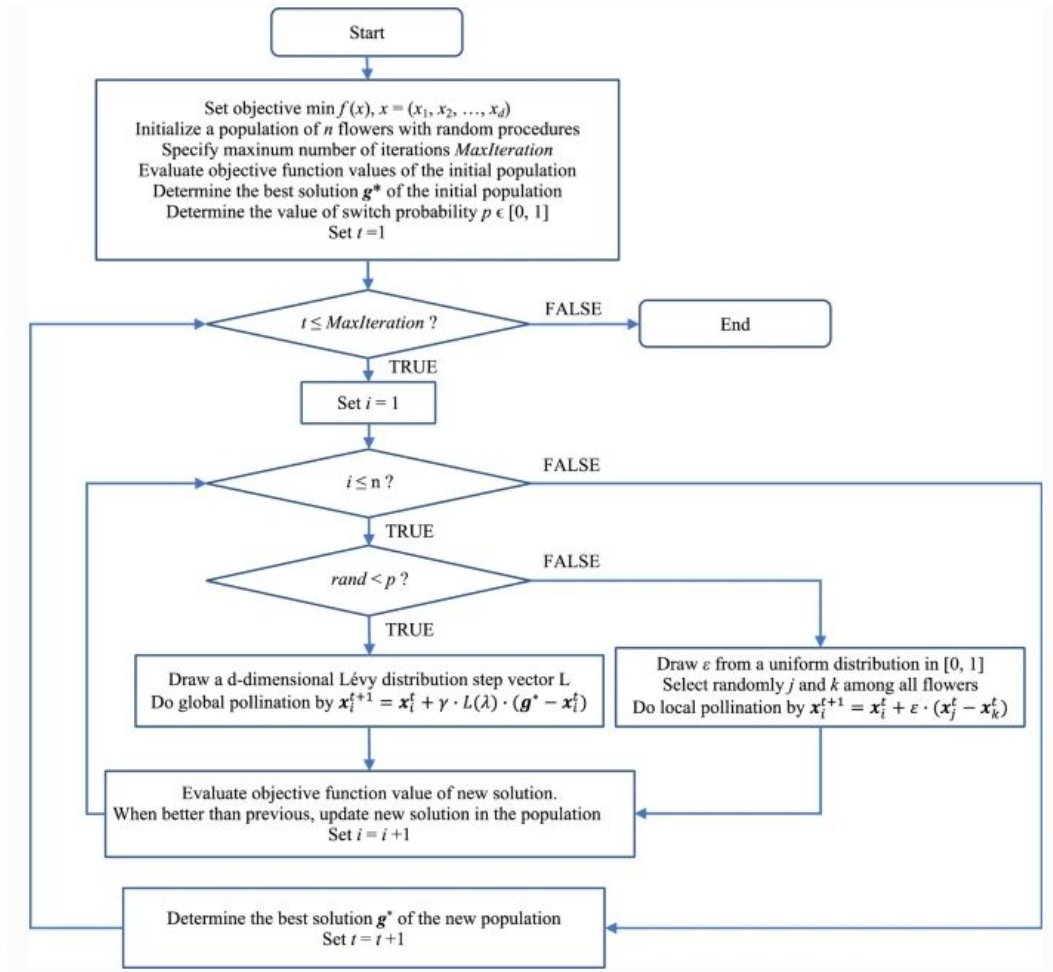


# Flower Pollination Algorithm

Piper Ellsworth, Jaehyun Lee, Shawn Hagler

# Methodology

# Data Flow Graph



# Python Code

```
In [3]: import random
import math
import os

def target_function():
    return

def initial_position(flowers = 3, min_values = [-5, -5], max_values = [5, 5], target_function = target_function):
    position = [[0] * (len(min_values) + 1) for _ in range(flowers)]

    for i in range(0, flowers):
        for j in range(0, len(min_values)):
            position[i][j] = random.uniform(min_values[j], max_values[j])
            position[i][-1] = target_function(position[i][0:len(min_values)])
    print(position)
    return position

def levy_flight(beta):
    beta = beta
    r1 = int.from_bytes(os.urandom(8), byteorder = "big") / ((1 << 64) - 1)
    r2 = int.from_bytes(os.urandom(8), byteorder = "big") / ((1 << 64) - 1)
    sig_num = math.gamma(1 + beta) * math.sin((math.pi * beta) / 2.0)
    sig_den = math.gamma((1 + beta) / 2) * beta * 2**((beta - 1) / 2)
    sigma = (sig_num / sig_den)**(1 / beta)
    levy = (0.01 * r1 * sigma) / (abs(r2)**(1 / beta))
    return levy

def clip(num, min_value, max_value):
    return max(min(num, max_value), min_value)
```

# Python Code

```
def pollination_global(position, best_global, flower = 0, gama = 0.5, lamb = 1.4, min_values = [-5, -5],
                       max_values = [5, 5], target_function = target_function):
    x = list(best_global)
    for j in range(0, len(min_values)):
        x[j] = clip((position[flower][j] + gama * levy_flight(lamb) * (position[flower][j] - best_global[j])),
                   min_values[j], max_values[j])

    x[-1] = target_function(x[0:len(min_values)])
    return x

def pollination_local(position, best_global, flower = 0, nb_flower_1 = 0, nb_flower_2 = 1, min_values = [-5,-5],
                      max_values = [5,5], target_function = target_function):
    x = list(best_global)
    for j in range(0, len(min_values)):
        r = int.from_bytes(os.urandom(8), byteorder = "big") / ((1 << 64) - 1)
        x[j] = clip((position[flower][j] + r * (position[nb_flower_1][j] - position[nb_flower_2][j])),
                   min_values[j], max_values[j])

    x[-1] = target_function(x[0:len(min_values)])
    return x
```

# Python Code

```
def flower_pollination_algorithm(flowers = 3, min_values = [-5, -5], max_values = [5, 5], iterations = 50,
                                gama = 0.5, lamb = 1.4, p = 0.8, target_function = target_function):

    count = 0
    score_list=[]
    position = initial_position(flowers = flowers, min_values = min_values, max_values = max_values,
                                target_function = target_function)

    best_global = sorted(position, key=lambda x: x[-1])[0]
    x = list(best_global)
    while (count <= iterations):
        print("Iteration = ", count, " f(x) = ", best_global[-1])
        score_list.append(best_global[-1])
        for i in range(0, len(position)):
            nb_flower_1 = int(random.random() * len(position))
            nb_flower_2 = int(random.random() * len(position))
            while nb_flower_1 == nb_flower_2:
                nb_flower_1 = int(random.random() * len(position))
            r = int.from_bytes(os.urandom(8), byteorder = "big") / ((1 << 64) - 1)
            if (r < p):
                x = pollination_global(position, best_global, flower = i, gama = gama, lamb = lamb,
                                        min_values = min_values, max_values = max_values, target_function = target_function)
            else:
                x = pollination_local(position, best_global, flower = i, nb_flower_1 = nb_flower_1,
                                        nb_flower_2 = nb_flower_2, min_values = min_values, max_values = max_values,
                                        target_function = target_function)

            if (x[-1] <= position[i][-1]):
                for j in range(0, len(x)):
                    position[i][j] = x[j]
        value = sorted(position, key=lambda x: x[-1])[0]
        if (best_global[-1] > value[-1]):
            best_global = list(value)
        count = count + 1
    print(best_global)
    return best_global, score_list
```



# Coding Alternatives

```
1 |
2 | package flower_pollination_package is
3 |     type real_vector is array(natural range <>) of real;
4 |     type real_vector_vector is array(natural range <>) of real_vector(1 to 3);
5 | end flower_pollination_package;
6 |
7 | library ieee;
8 | use ieee.std_logic_1164.all;
9 | use ieee.std_logic_unsigned.all;
10 | use ieee.numeric_std.all;
11 | use ieee.math_real.all;
12 | use work.flower_pollination_package.all;
13 |
14 | entity flower_pollination is
15 |     port (
16 |         clk: in std_logic;
17 |         reset: in std_logic;
18 |         flower_count: in integer;
19 |         min: in integer;
20 |         max: in integer;
21 |         gamma: in real;
22 |         lamb: in real;
23 |         p: in real;
24 |         iterations: in integer;
25 |         best_solution: out real_vector(1 to 3)
26 |     );
27 | end flower_pollination;
28 |
29 | architecture beh of flower_pollination is
30 |     function random_real return real_vector is
31 |         variable seed1: integer := 123456789;
32 |         variable seed2: integer := 987654321;
33 |         variable seed3: integer := 239438458;
34 |         variable rand_real1: real;
35 |         variable rand_real2: real;
36 |         variable random: real_vector(1 to 2);
37 |     begin
38 |         uniform(seed1, seed2, rand_real1);
39 |         uniform(seed2, seed3, rand_real2);
40 |         random(1) := rand_real1;
41 |         random(2) := rand_real2;
42 |         return random;
43 |     end random_real;
44 |
```



# Coding Alternatives

```
44 |
45 | function random_integer(min_value: integer; max_value: integer) return integer is
46 |     variable real_random: real;
47 |     variable seed1, seed2: positive := 987654321;
48 | begin
49 |     uniform(seed1, seed2, real_random);
50 |     return integer(real(min_value) + real_random * real(max_value - min_value + 1));
51 | end random_integer;
52 |
53 | function six_hump_camel_back(variables: real_vector) return real is
54 |     variable x: real := variables(1);
55 |     variable y: real := variables(2);
56 | begin
57 |     return 4.0 * (x**2) - 2.1 * (x**4) + ((1.0/3.0) * (x**6)) + x * y - 4.0 * (y**2) + 4.0 * (y**4);
58 | end six_hump_camel_back;
59 |
60 | function fitness_compare(a: real_vector; b: real_vector) return boolean is
61 | begin
62 |     return (six_hump_camel_back(a) <= six_hump_camel_back(b));
63 | end fitness_compare;
64 |
65 | function initial_positions(flower_count: integer; min: integer; max: integer) return real_vector_vector is
66 |     variable positions: real_vector_vector(1 to flower_count);
67 | begin
68 |     for i in 1 to flower_count - 1 loop
69 |         for j in 1 to 2 loop
70 |             positions(i)(j) := real(min) + random_real(j) * real(max - min + 1);
71 |         end loop;
72 |         positions(i)(3) := six_hump_camel_back(positions(i));
73 |     end loop;
74 |     return positions;
75 | end initial_positions;
```

# Coding Alternatives

```
function pollination_global(positions: real_vector_vector; best_position: real_vector; min: integer; max: integer; flower: integer; gamma: real; lamb: real) return real_vector is
  variable x: real_vector(1 to 3);
  variable delta: real;
begin
  for i in 1 to 2 loop
    x(i) := positions(flower)(i) + gamma * levy_flight(lamb) * (best_position(i) - positions(flower)(i));
    if x(i) < real(min) then
      x(i) := real(min);
    elsif x(i) > real(max) then
      x(i) := real(max);
    end if;
  end loop;
  end loop;
  x(3) := six_hump_camel_back(x);
  return x;
end pollination_global;
```

```
function pollination_local(flower_count: integer; flower: integer; positions: real_vector_vector; min: integer; max: integer) return real_vector is
  variable x: real_vector(1 to 3);
  variable delta: real;
  variable r: real;
  variable nb_flower_1: integer := random_integer(1, flower_count);
  variable nb_flower_2: integer := random_integer(1, flower_count);
begin
  r := random_real(1);
  for i in 1 to 2 loop
    delta := r * (positions(nb_flower_1)(i) - positions(nb_flower_2)(i));
    if (positions(flower)(i) + delta > real(max) then
      x(i) := real(max);
    elsif (positions(flower)(i) + delta < real(min) then
      x(i) := real(min);
    else
      x(i) := positions(flower)(i) + delta;
    end if;
  end loop;
  x(3) := six_hump_camel_back(x);
  return x;
end pollination_local;
```

# Coding Alternatives

```
128 :
129 :     signal count: integer := 0;
130 : begin
131 :     process (clk, reset)
132 :         variable positions: real_vector_vector(1 to 175);
133 :         variable best_position: real_vector(1 to 3);
134 :         variable x: real_vector(1 to 3);
135 :     begin
136 :         if (reset = '1') then
137 :             count <= 0;
138 :             positions := initial_positions(flower_count, min, max);
139 :             best_position := positions(1);
140 :         elsif (rising_edge(clk)) then
141 :             if (count < iterations) then
142 :                 for i in 1 to flower_count loop
143 :                     if (random_real(1) < p) then
144 :                         x := pollination_global(positions, best_position, min, max, i, gamma, lamb);
145 :                     else
146 :                         x := pollination_local(flower_count, i, positions, min, max);
147 :                     end if;
148 :                     if (fitness_compare(positions(i), best_position)) then
149 :                         best_position := positions(i);
150 :                     end if;
151 :                 end loop;
152 :                 count <= count + 1;
153 :             end if;
154 :         else
155 :             best_solution <= best_position;
156 :         end if;
157 :     end process;
158 : end beh;
```

# Limitations

1. Complex algorithm design: The FPA involves complex mathematical operations. Translating these intricate operations into VHDL code may require significant effort and a thorough understanding of both the algorithm and the hardware design.
2. Limited resources: FPGA devices have limited resources, including logic elements, memory, and DSP blocks.
3. Performance constraints: FPGA based implementations of the FPA may face performance limitations due to the inherent parallelism and pipeline capabilities of FPGAs.
4. Debugging and testing complexities: Debugging and testing FPGA designs is inherently more challenging than software-based implementations.

# Division of VHDL code

Lead Software Developer: Shawn

Assistant Software Developer: Jason

Tester/Debugger: Piper

# Results

# VHDL Code- Main

```
80      begin
81      process (clk, rst)
82          variable xx : real_vector(1 to 2);
83          variable temp, minimum : real := 0.0;
84          variable best_vector : real_vector(1 to 2) := (others => 0.0);
85          variable count : integer;
86      begin
87          if (rst = '1') then
88              best_vector := (others => 0.0);
89              minimum := 0.0;
90          elsif (rising_edge(clk)) then
91              if (xxx < p) then
92                  xx := pollination_global(x_vector, levy_vector, best_vector, min, max, gamma, lamb);
93              else
94                  xx := pollination_local(local_vector, x_vector, xxx, min, max);
95              end if;
96              temp := ((4.0 - 2.1*(xx(1)**2) + (xx(1)**4)/3.0))*(xx(1)**2)
97                  + (xx(1)*xx(2) + (-4.0 + 4.0*(xx(2)**2))*(xx(2)**2));
98              if (temp <= minimum) then
99                  minimum := temp;
100                 best_vector(1) := xx(1);
101                 best_vector(2) := xx(2);
102             end if;
103
104             best_x(1) <= best_vector(1);
105             best_x(2) <= best_vector(2);
106             fx <= minimum;
107         end if;
108     end process;
109 end Behavioral;
```

# VHDL Code- Global

```
28 architecture Behavioral of final2 is
29
30     function levy_flight(beta: real; levy_vector: real_vector) return real is
31         constant sig_num: real := 0.9399856029866254;
32         constant sig_den: real := 1.6168504121556964;
33         variable sigma: real;
34         variable levy: real;
35         variable r1, r2: real;
36     begin
37         r1 := levy_vector(1);
38         r2 := levy_vector(2);
39         sigma := (sig_num / sig_den) ** (1.0 / beta);
40         levy := (0.01 * r1 * sigma) / (real(abs(r2)) ** (1.0 / beta));
41         return levy;
42     end levy_flight;
43
44     function pollination_global(position: real_vector; levy_vector: real_vector; best_position: real_vector;
45                               min: integer; max: integer; gamma: real; lamb: real) return real_vector is
46         variable x: real_vector(1 to 2);
47         variable delta: real;
48     begin
49         for i in 1 to 2 loop
50             x(i) := position(i) + gamma * levy_flight(lamb, levy_vector) * (best_position(i) - position(i));
51             if x(i) < real(min) then
52                 x(i) := real(min);
53             elsif x(i) > real(max) then
54                 x(i) := real(max);
55             end if;
56         end loop;
57         return x;
58     end pollination_global;
59
```

$$L \sim \frac{(\beta + 1) \times \sin\left(\frac{\beta\pi}{2}\right)}{\pi} \times \frac{1}{s^{\beta+1}}, \quad (s \gg s_0 > 0)$$

$$x_i^{t+1} = x_i^t + L(x_i^t - g^*)$$



# VHDL Code- Local

```
60 function pollination_local(local_vector: real_vector; position: real_vector; xxx: real; min: integer;
61                                     max: integer) return real_vector is
62     variable x: real_vector(1 to 2);
63     variable delta: real;
64     variable r: real;
65 begin
66     r := xxx;
67     for i in 1 to 2 loop
68         delta := r * (position(i) - local_vector(i));
69         if (position(i) + delta) > real(max) then
70             x(i) := real(max);
71         elsif (position(i) + delta) < real(min) then
72             x(i) := real(min);
73         else
74             x(i) := position(i) + delta;
75         end if;
76     end loop;
77     return x;
78 end pollination_local;
```

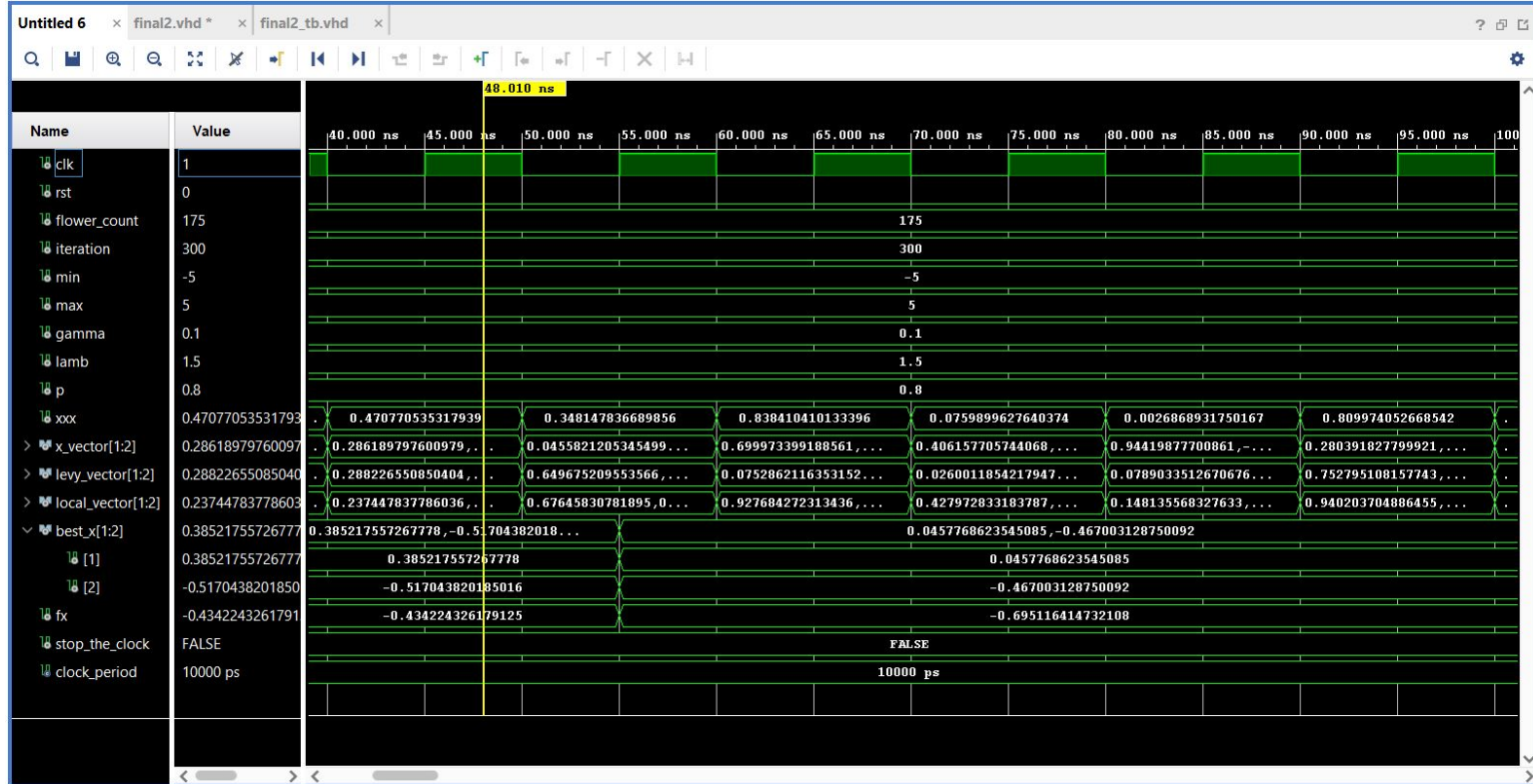
$$x_i^{t+1} = x_i^t + \epsilon (x_j^t - x_k^t)$$

# Testbench

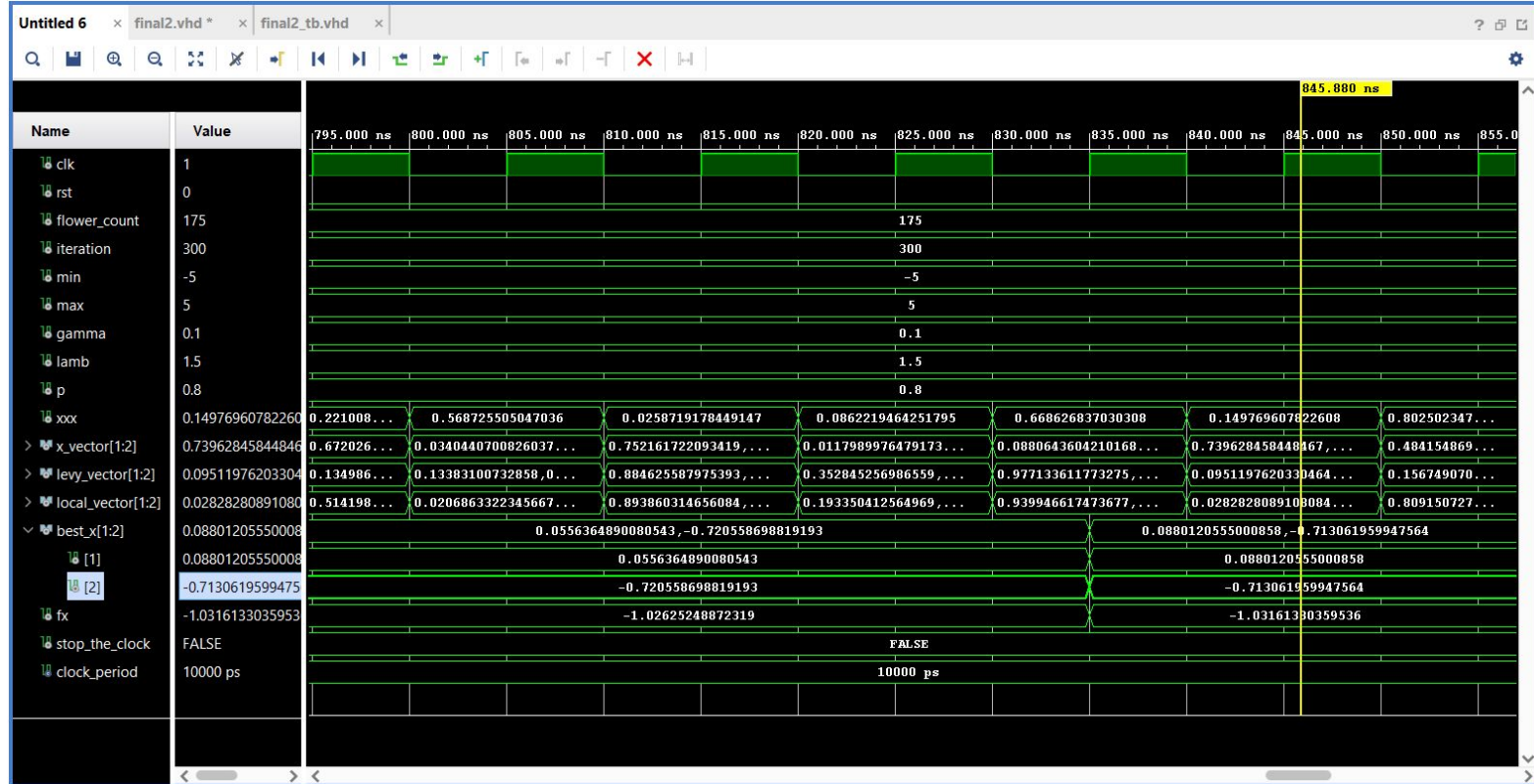
```
106 clocking: process
107 begin
108     while not stop_the_clock loop
109         clk <= '0', '1' after clock_period / 2;
110         wait for clock_period;
111     end loop;
112     wait;
113 end process;
```

```
65 stimulus: process
66     variable seed1, seed2, seed3, seed4, seed5 : positive;
67     variable xx, y, z, levy1, levy2, local1, local2 : real;
68 begin
69     rst <= '1';
70     wait for 10 ns;
71     rst <= '0';
72     flower_count <= 175;
73     iteration <= 300;
74     min <= -5;
75     max <= 5;
76     gamma <= 0.1;
77     lamb <= 1.5;
78     p <= 0.8;
79
80     seed1 := 1;
81     seed2 := 2;
82     seed3 := 3;
83     seed4 := 4;
84     seed5 := 5;
85     for n in 1 to 300 loop
86         uniform(seed1, seed2, xx);
87         uniform(seed1, seed3, y);
88         uniform(seed2, seed3, z);
89         uniform(seed1, seed4, levy1);
90         uniform(seed1, seed5, levy2);
91         uniform(seed2, seed4, local1);
92         uniform(seed3, seed5, local2);
93         xxx <= xx;
94         x_vector(1) <= y-0.002;
95         x_vector(2) <= z-0.984;
96         levy_vector(1) <= levy1;
97         levy_vector(2) <= levy2;
98         local_vector(1) <= local1;
99         local_vector(2) <= local2;
100         wait for 10 ns;
101     end loop;
102
103     wait;
104 end process;
```

# Simulation Waveform



# Simulation Waveform

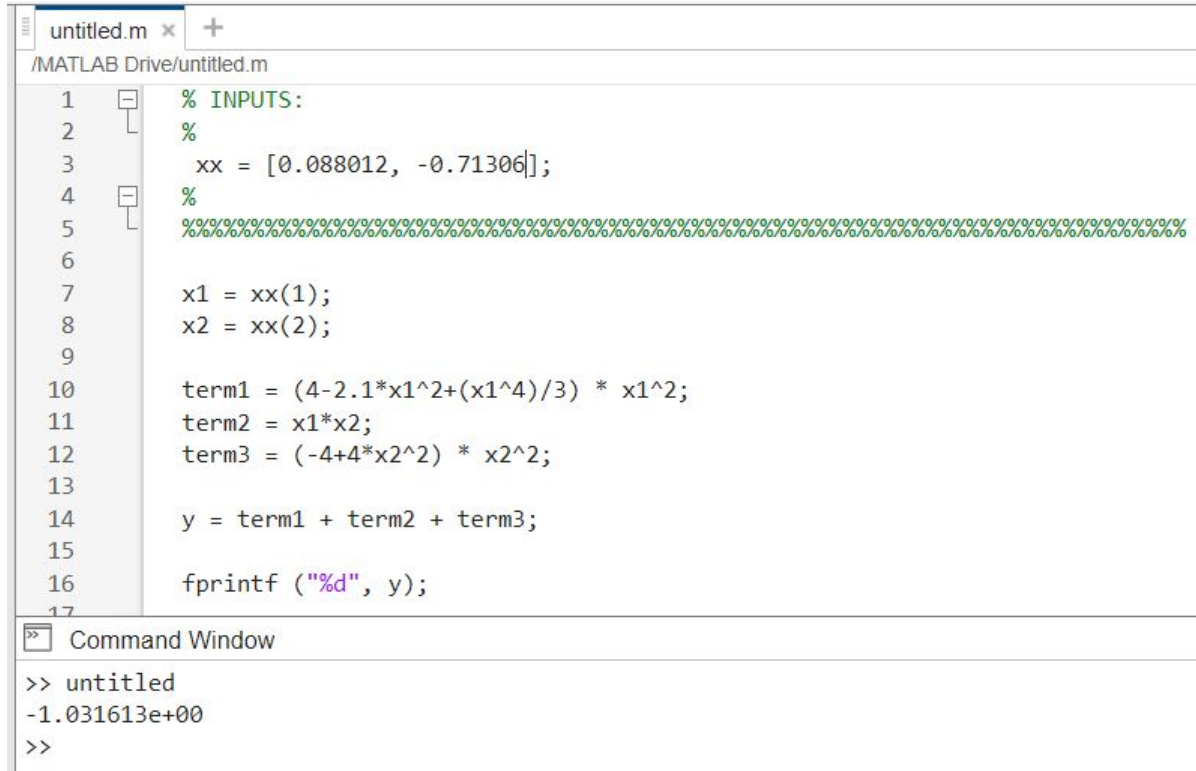


# Calculations Python

```
def six_hump_camel_back(variables_values = [0, 0]):  
    return 4 * variables_values[0]**2 - 2.1 * variables_values[0]**4 + (1/3) * variables_values[0]**6  
        + variables_values[0] * variables_values[1] - 4 * variables_values[1]**2 + 4 * variables_values[1]**4  
  
flower_best_solution, score_list = flower_pollination_algorithm(flowers = 175, min_values = [-5,-5],  
                                                                max_values = [5,5], iterations = 300,  
                                                                gama = 0.1, lamb = 1.5, p = 0.8,  
                                                                target_function = six_hump_camel_back)
```

```
Iteration = 283 f(x) = -1.0316275783783373  
Iteration = 284 f(x) = -1.0316275783783373  
Iteration = 285 f(x) = -1.0316275783783373  
Iteration = 286 f(x) = -1.0316275783783373  
Iteration = 287 f(x) = -1.0316275783783373  
Iteration = 288 f(x) = -1.0316275783783373  
Iteration = 289 f(x) = -1.0316275783783373  
Iteration = 290 f(x) = -1.0316275783783373  
Iteration = 291 f(x) = -1.0316275783783373  
Iteration = 292 f(x) = -1.0316275783783373  
Iteration = 293 f(x) = -1.0316275783783373  
Iteration = 294 f(x) = -1.0316275783783373  
Iteration = 295 f(x) = -1.0316275783783373  
Iteration = 296 f(x) = -1.0316275783783373  
Iteration = 297 f(x) = -1.0316275783783373  
Iteration = 298 f(x) = -1.0316275783783373  
Iteration = 299 f(x) = -1.0316275783783373  
Iteration = 300 f(x) = -1.0316275783783373  
[0.08945476939947151, -0.7128225465183462, -1.0316275783783373]
```

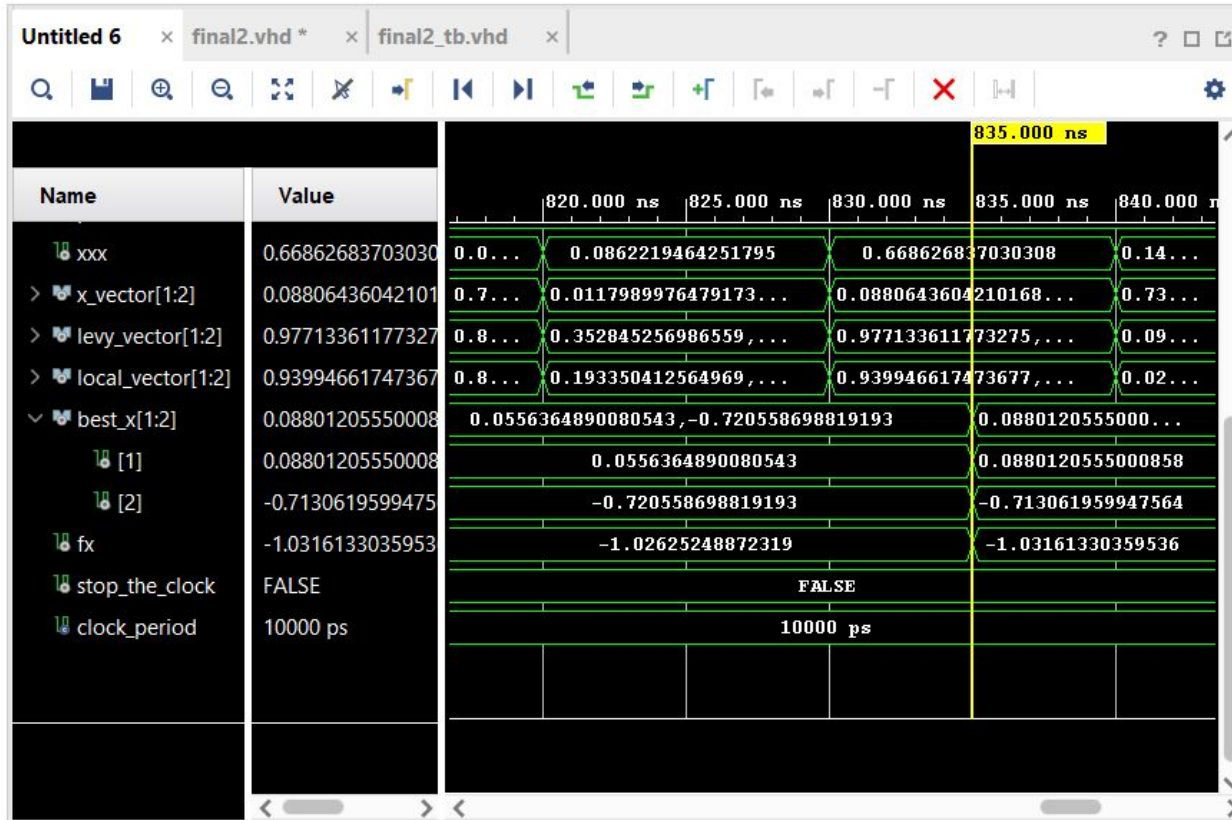
# Calculations MATLAB



The image shows a MATLAB editor window with a script named 'untitled.m' and a Command Window. The script defines a vector 'xx' and calculates three terms based on its elements, then sums them to produce 'y'.

```
untitled.m x +  
/MATLAB Drive/untitled.m  
1 % INPUTS:  
2 %  
3 xx = [0.088012, -0.71306];  
4 %  
5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
7 x1 = xx(1);  
8 x2 = xx(2);  
9  
10 term1 = (4-2.1*x1^2+(x1^4)/3) * x1^2;  
11 term2 = x1*x2;  
12 term3 = (-4+4*x2^2) * x2^2;  
13  
14 y = term1 + term2 + term3;  
15  
16 fprintf ("%d", y);  
17  
Command Window  
>> untitled  
-1.031613e+00  
>>
```

# Timing Information of Software Solution



# Timing Information of Software Solution Python

```
start = time.time()
...
end = time.time()
print(end - start)
```

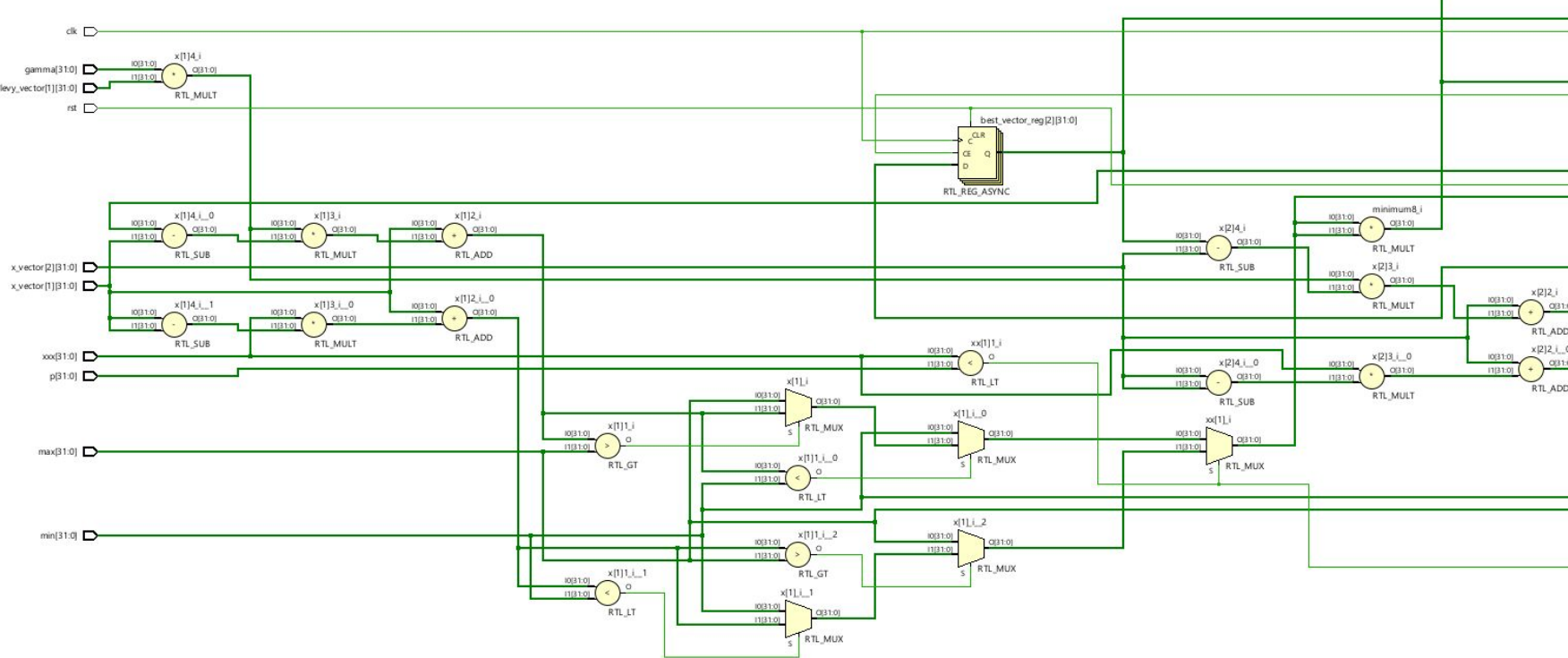
```
Iteration = 181 f(x) = -1.0315827026212248
Iteration = 182 f(x) = -1.0315827026212248
Iteration = 183 f(x) = -1.0315827026212248
Iteration = 184 f(x) = -1.0315827026212248
time elapsed: 1.0451202392578125
Iteration = 185 f(x) = -1.0315853648975715
time elapsed: 1.05312180519104

Iteration = 186 f(x) = -1.0316095330641744
Iteration = 187 f(x) = -1.0316095330641744
Iteration = 188 f(x) = -1.0316095330641744
Iteration = 189 f(x) = -1.0316095330641744
Iteration = 190 f(x) = -1.0316095330641744
Iteration = 191 f(x) = -1.0316095330641744
time elapsed: 1.0831239223480225
Iteration = 192 f(x) = -1.031617853236341
Iteration = 193 f(x) = -1.031617853236341
Iteration = 194 f(x) = -1.031617853236341
Iteration = 195 f(x) = -1.031617853236341
```

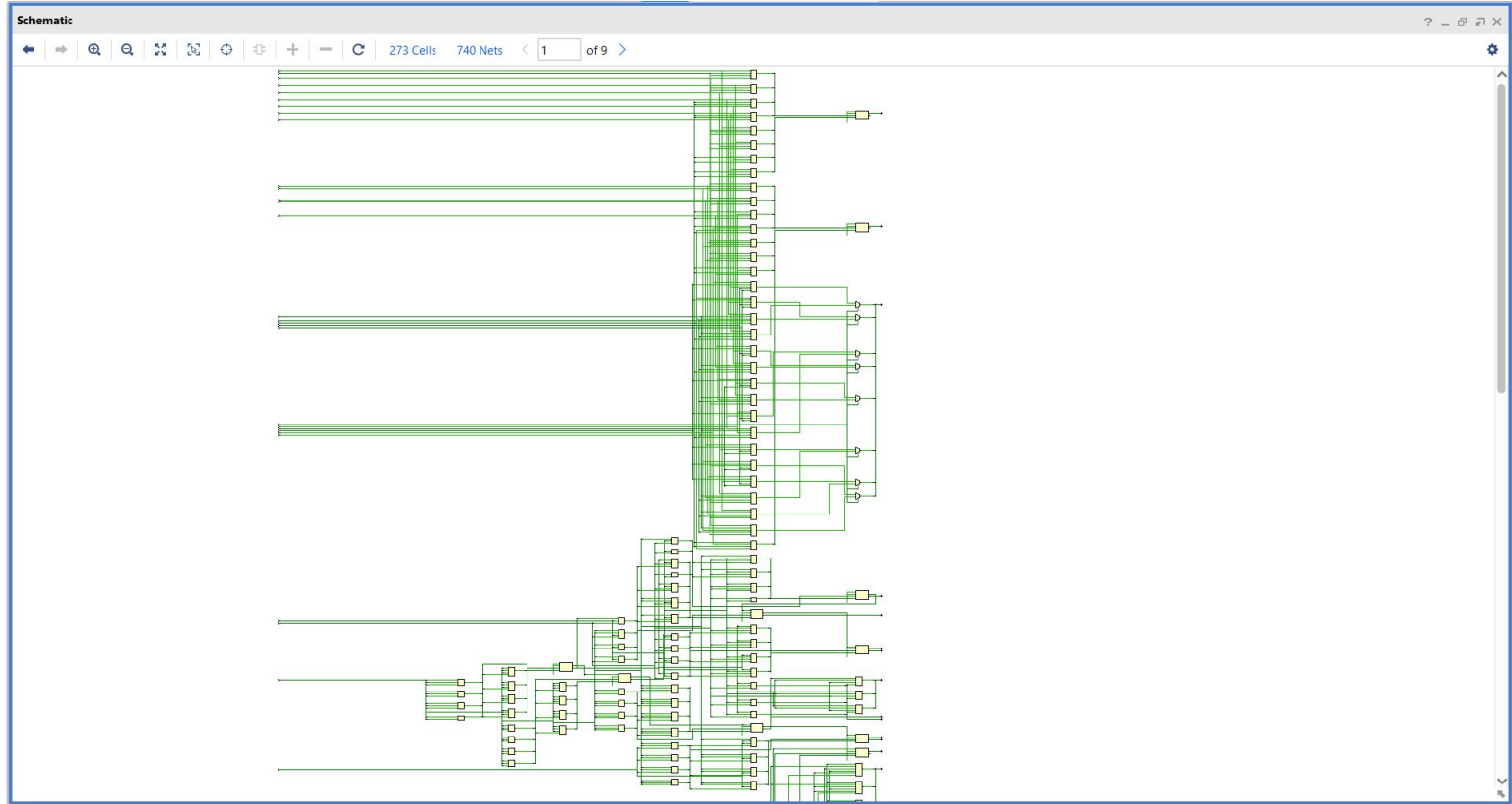




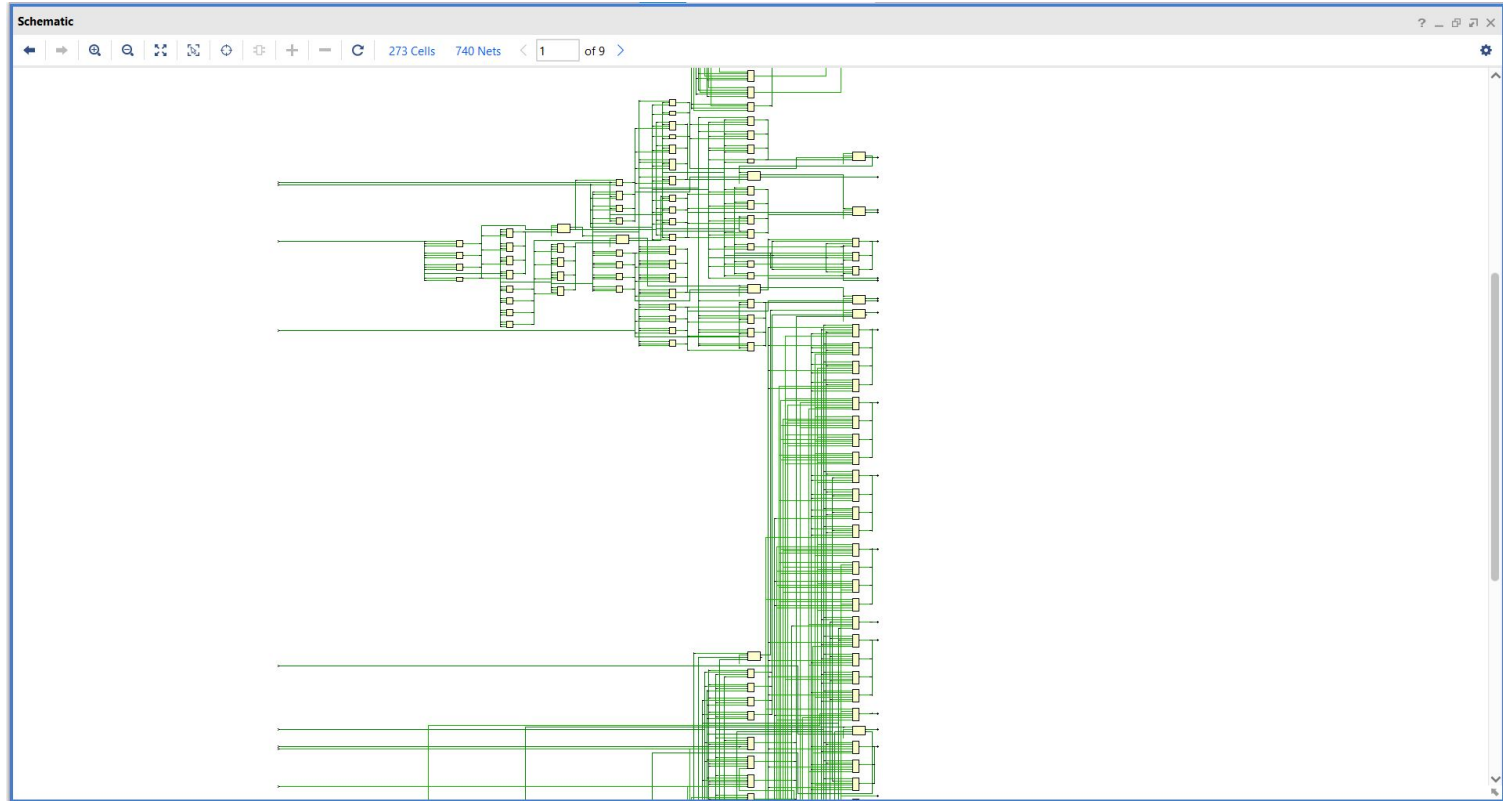
# RTL Analysis



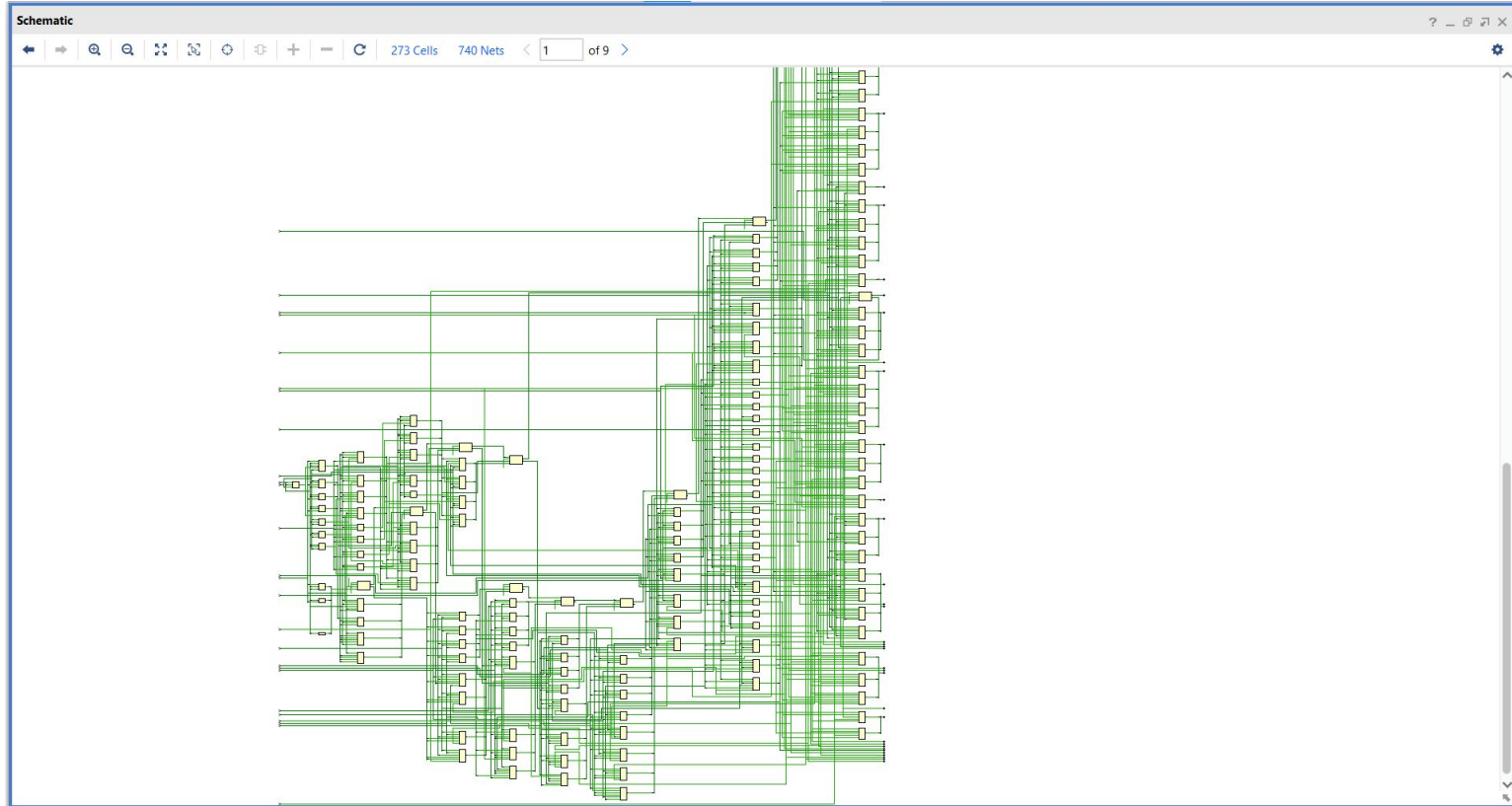
# Synthesized Circuit



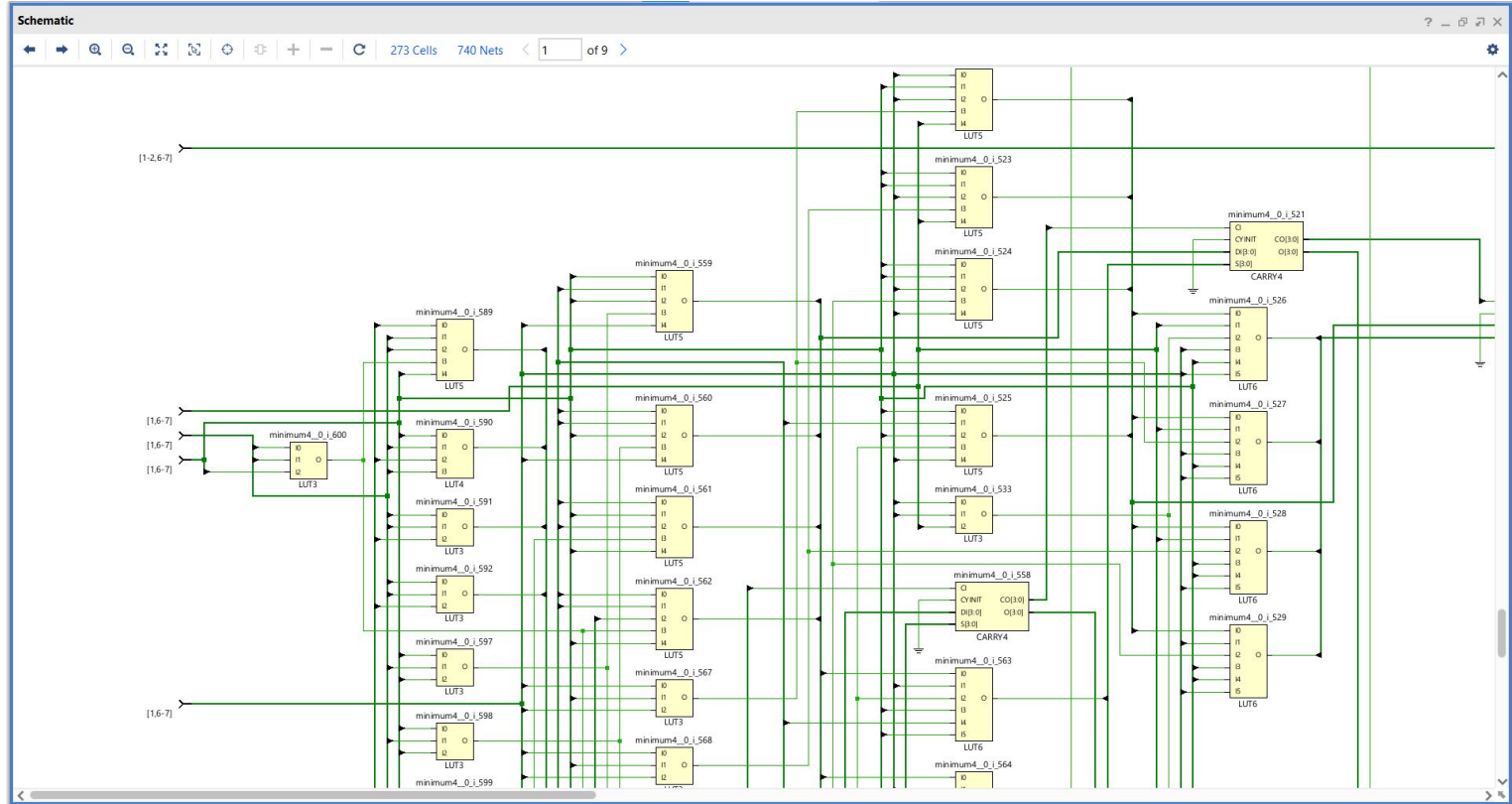
# Synthesized Circuit



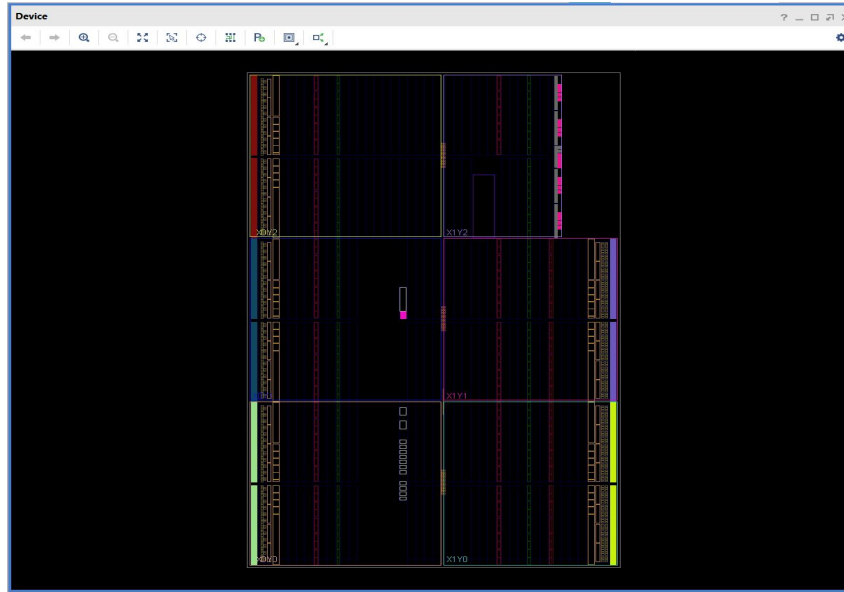
# Synthesized Circuit



# Synthesized Circuit



# Device Implementation



## Place Design (103 errors)

- ❗ [Place 30-415] IO Placement failed due to overutilization. This design contains 354 I/O ports while the target device: 7a35t package: cpg236, contains only 106 available user I/O. The target device has 106 usable I/O pins of which 0 are already occupied by user-locked I/Os. To rectify this issue:
  1. Ensure you are targeting the correct device and package. Select a larger device or different package if necessary.
  2. Check the top-level ports of the design to ensure the correct number of ports are specified.
  3. Consider design changes to reduce the number of I/Os necessary.

# Timing Analysis

The screenshot shows a software interface with a menu bar at the top containing 'Tcl Console', 'Messages', 'Log', 'Reports', 'Design Runs', 'Timing', and 'Linter'. Below the menu bar is a toolbar with icons for search, zoom, refresh, and save. The main content area is titled 'Design Timing Summary' and is divided into three columns: 'Setup', 'Hold', and 'Pulse Width'. The 'Setup' column shows Worst Negative Slack (WNS) at -53.740 ns, Total Negative Slack (TNS) at -5585.348 ns, Number of Failing Endpoints at 161, and Total Number of Endpoints at 225. The 'Hold' column shows Worst Hold Slack (WHS) at 0.158 ns, Total Hold Slack (THS) at 0.000 ns, Number of Failing Endpoints at 0, and Total Number of Endpoints at 225. The 'Pulse Width' column shows Worst Pulse Width Slack (WPWS) at 4.500 ns, Total Pulse Width Negative Slack (TPWS) at 0.000 ns, Number of Failing Endpoints at 0, and Total Number of Endpoints at 162. A red warning icon is present in the top left of the main area, and a red warning icon is also in the left sidebar next to 'Design Timing Summary'. At the bottom of the main area, a red text message states 'Timing constraints are not met.' The left sidebar contains a tree view with categories: 'General Information', 'Timer Settings', 'Design Timing Summary' (selected), 'Clock Summary (1)', 'Methodology Summary', 'Check Timing (353)', 'Intra-Clock Paths', 'Inter-Clock Paths', and 'Other Path Groups'. A status bar at the bottom left of the window displays 'Timing Summary - timing\_1'.

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): -53.740 ns	Worst Hold Slack (WHS): 0.158 ns	Worst Pulse Width Slack (WPWS): 4.500 ns
Total Negative Slack (TNS): -5585.348 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 161	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 225	Total Number of Endpoints: 225	Total Number of Endpoints: 162

**Timing constraints are not met.**

Timing Summary - timing\_1



# Timing Analysis

Timing

Timing Checks - Setup

General Information

Settings

Timing Checks (20)

Setup (10)

Hold (10)

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Logic %	Net %	Requirement	Source Clock	Destination Clock	Exception
Path 1	-53.740	68	37	96	best_vector_reg[1][1]/C	best_vector_reg[1][0]/CE	63.358	40.651	22.707	64.2	35.8	10.000	clk	clk	
Path 2	-53.740	68	37	96	best_vector_reg[1][1]/C	best_vector_reg[1][10]/CE	63.358	40.651	22.707	64.2	35.8	10.000	clk	clk	
Path 3	-53.740	68	37	96	best_vector_reg[1][1]/C	best_vector_reg[1][11]/CE	63.358	40.651	22.707	64.2	35.8	10.000	clk	clk	
Path 4	-53.740	68	37	96	best_vector_reg[1][1]/C	best_vector_reg[1][12]/CE	63.358	40.651	22.707	64.2	35.8	10.000	clk	clk	
Path 5	-53.740	68	37	96	best_vector_reg[1][1]/C	best_vector_reg[1][13]/CE	63.358	40.651	22.707	64.2	35.8	10.000	clk	clk	
Path 6	-53.740	68	37	96	best_vector_reg[1][1]/C	best_vector_reg[1][14]/CE	63.358	40.651	22.707	64.2	35.8	10.000	clk	clk	
Path 7	-53.740	68	37	96	best_vector_reg[1][1]/C	best_vector_reg[1][15]/CE	63.358	40.651	22.707	64.2	35.8	10.000	clk	clk	
Path 8	-53.740	68	37	96	best_vector_reg[1][1]/C	best_vector_reg[1][16]/CE	63.358	40.651	22.707	64.2	35.8	10.000	clk	clk	
Path 9	-53.740	68	37	96	best_vector_reg[1][1]/C	best_vector_reg[1][17]/CE	63.358	40.651	22.707	64.2	35.8	10.000	clk	clk	
Path 10	-53.740	68	37	96	best_vector_reg[1][1]/C	best_vector_reg[1][18]/CE	63.358	40.651	22.707	64.2	35.8	10.000	clk	clk	

# Power Report

