EE8219 Field Programmable Gate Array Architectures
Final Course Project: Simulated Annealing Program.

Spiros Hippolyte, Prof. Andy Ye, PhD.
Ryerson University. Department. Electrical Computer Engineering.

1. Introduction

This report details the implementation of the simulated annealing placement algorithm in the C++ coding language. Its purpose is to simulate the placement of logic blocks with the FPGA, thus providing a specific CAD tool. The following sections will describe the program design, results & analysis, primary program listing and code.

2. Program Design

The program is structured according to the algorithm of simulated annealing, applied to the placement of FPGA logic blocks (composed of LUT, D-FF and 2:1 MUX). The program uses a class object oriented programing to provide data encapsulation. The main function of the program provides the user command line interface, providing a selection of 12 benchmark circuit files. The file handling is passed to the class "Simulated Annealing" that first reads the file data per line (reading in two sets of logic block data) and passes the encoded logic block (x,y co-ordinate & pin data). The x,y co-ordinates are compared first to see if a set on the same line are equal (representing the same net) then current and past sets are compared for duplicate entries. Storage of block data is performed via the vector structure (vector<LogicBlock> LBlock) that holds the co-ordinates and pin data of blocks within a structure object (struct LogicBlock). A count of the number of nets inputted is also held within the structure. (Operations on the vectors are scaled to their size).

The program now starts the initialization process starting next with the, RandomPlacement member function is called to reassign existing block locations with pseudo randomly generated ones (using the system clock as a seed value).
A comparison between random blocks is also performed. If blocks are repeated, then random co-ordinate values are regenerated for them. The net list is also updated with its own vector (also performed in the List function). The Initial Temperature function is called. (Note that the temperature variable is initialized to 1000 to simulate a realistic annealing process).
Its requirements are as follows:

- Randomly place all logic cluster.
- Perform N moves to obtain N configurations.
- Compute the standard deviation of the cost of N configurations.
- Compute the new temperature=20 x σ.

The first criteria is performed by the RandomPlacement function; within the InitalTemp function the Cost function is run in a loop for N iterations, also summing the resultant semi-perimeter value "costwire". This data is used to compute the standard deviation (from the variance, mean computed prior). The next function computes the initial range of move "RangeMove".
It computes the range based on the formula (1):

$$(1)\quad R=R(1-0.44+\alpha)$$

Where the first input range is the array dimension "r" (row/column size of the FPGA).
The acceptance rate α is set to 0.8 (default for VPR) which is within the optimal range for temperature updating.

With the initial conditions set, the program enters its core phase within the while loop, under the condition that the temperature is greater than zero. Setting an initial high temperature allows for greater risk and distance moves. An inner while checks if the ratio of moves per temperature has not been reached as compared to a loop counter. The MovePerTemp function (2) computes the ratio as the following.

(2) MovePerTemp=Constant (set to 1.0) x $N^{4/3}$

While these conditions have not been met, the Move function is called where previously randomly placed blocks are re-assigned its location with the range calculation, producing new block placements. The difference in the cost of block moves, ΔC is computed were past semi-perimeter cost and current ones are evaluated. If the old cost B is less the new cost A, then the difference of B-A is computed and vice versa (A>B then A-B). This method is used to obtain the optimal, minimum cost, while each cost is evaluated till the global minimum is reached. A randomly generated number ("rm" in the program), (set to normal distribution to produce values from 0 to 1) is compared to the exponential of ΔC/T, as the equation below describes (3).

(3) $r < e^{-\Delta C/T}$

Where if true a new set of randomly generated block placements are performed via the RandomPlacement function. The temperature and range of move are recalculated via the TempUpdate and the RangeMove functions respectively. The, TempUpdate function goal is to adjust the temperature with respect to the acceptance rate α. The updated temperature is the product of the current and the update factor γ, which is effected by the range of α. This final determination is also dependent on the error factor ε (set to 0.005) and the ratio of costs per nets. The following presents the equations for updated temperature (table 1).

$0.8 > 0.96$	γ=0.5
$0.8 \leq \alpha \leq 0.96$	γ=0.9
$0.15 \leq \alpha \leq 0.8$	**γ=0.95**
$\alpha < 0.15$	γ=0.8

Table 1. Temperature Update computation table.

The optimal range for α is between 0.15 and 0.8, producing the largest factor =0.95 in bold. After the multiplication factor has been selected, the key error factor ε is now computed. The DrawFPGA & DrawRoutine functions initialize the graphics environment (the project program was developed in the Windows environment) and then compute the position and render the logic blocks, shading in placed logic blocks respectively.

Results & Analysis

The resulting program output produced a zero cost result for all benchmark circuits, producing a number of moves results equal to the array size of each circuit, which would produce a flat graph. The twelve resultant output files are labeled "Placement fcct#-#.txt" and are included with this project submission. These results are not consistent with the algorithm. The output also include the random placements of the first five benchmark circuits (fig 1 – 5).

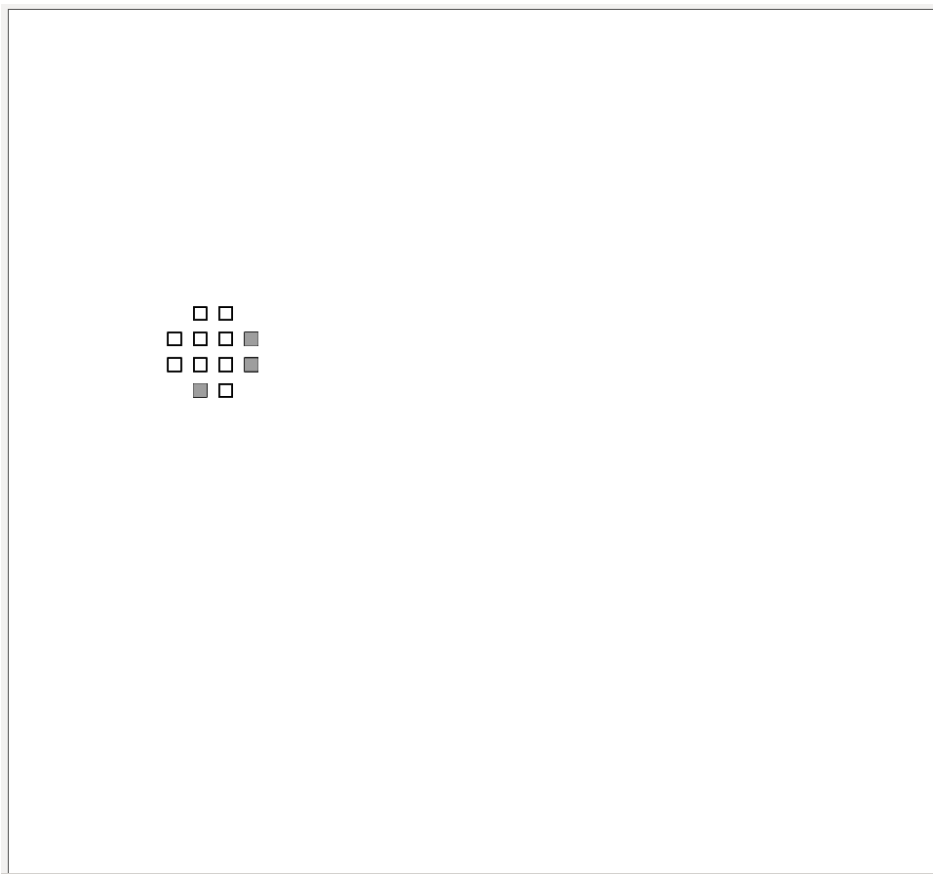

Fig 1. Circuit, fcct1.

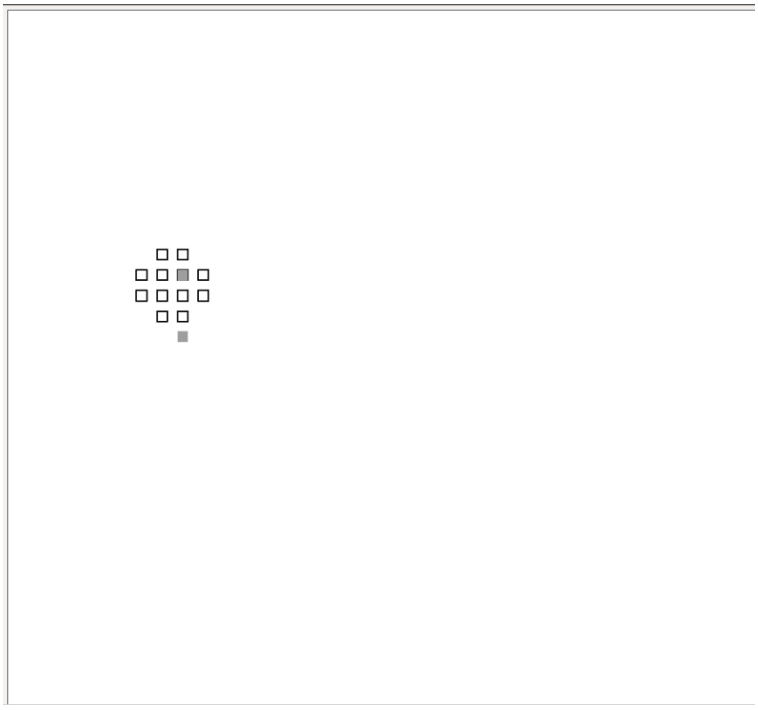

Fig. 2. Circuit: fcct1-1.

A. Debugging

Originally in the program, circuit data was read into the main function, it was later moved with the Simulated Annealing class. A function called List would then handle the input, but code debugging shown that vector data was not being retained; therefore, file input code was placed directly in the while loop (no function call) to provide direct data transfer. Results of the random placement function show the new co-ordinates on the command line. The DrawFPGA function can be called within or after the random placement function, yet beyond the fifth file with array size of 5 or greater, produces a runtime error (either with the program or from the "graphics.cpp" file functions). When the DrawFPGA function (that also calls the DrawRoutine function) is working, it produces the PS file via direct function call to the PS function. Debugging show the graphical placements of random block locations, as do the PS file outputs screen shots from Linux Ubuntu. Further debugging of the code would focus on creating net groupings and to computer there semi-perimeter costs in order to produce a variance needed to obtain the standard deviation required to properly update the temperature, allowing the program in enter the main functional loop. A full examination of the data passing between all vectors and variables and there real time order of operations will be required.

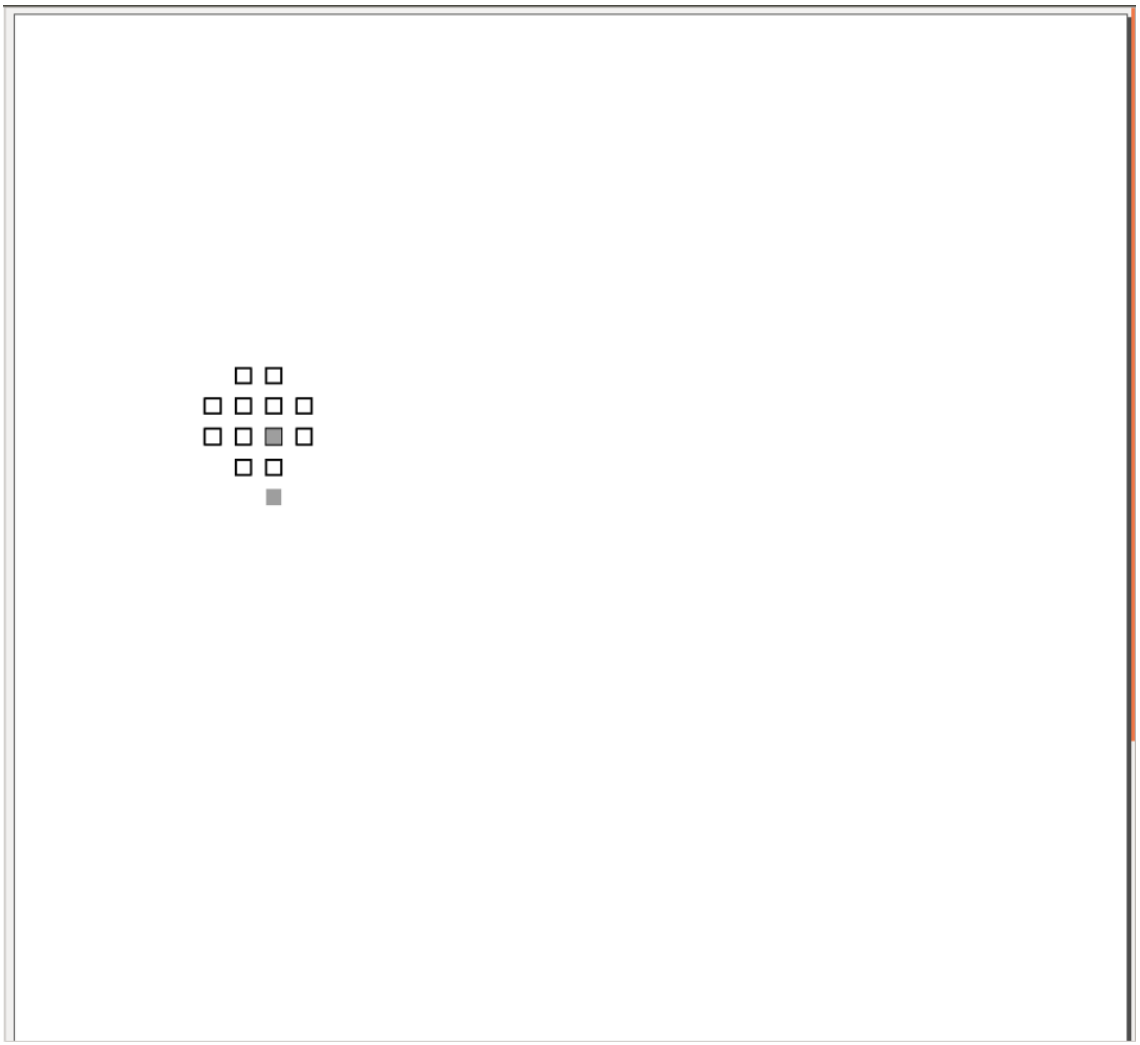

Fig 3. Circuit: fcct1-2.

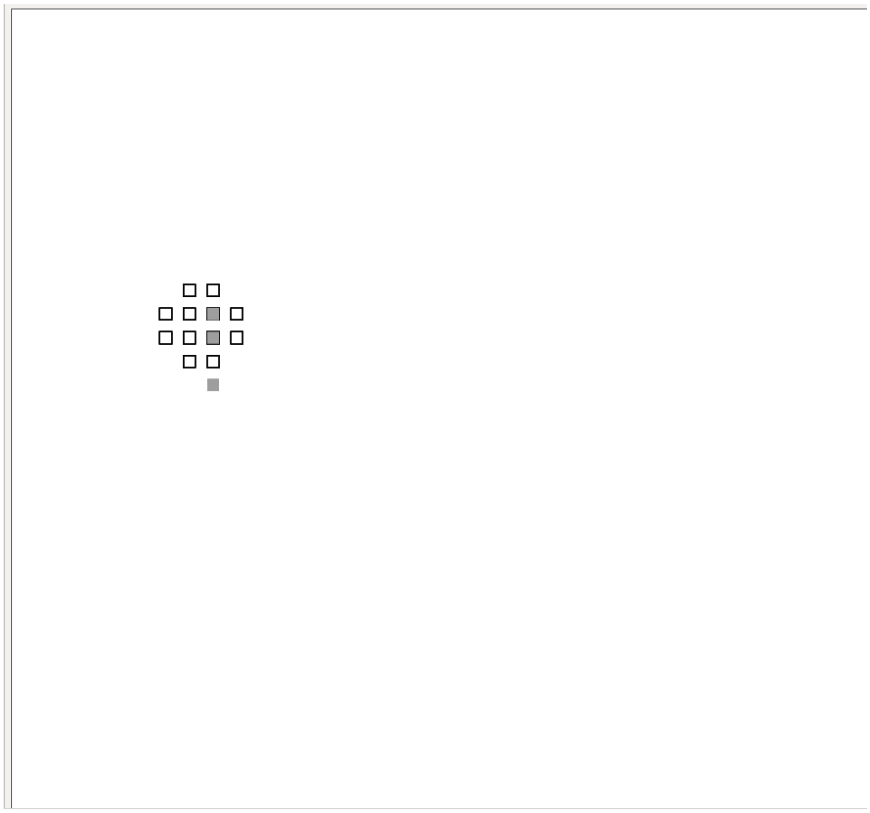

Fig 4. Circuit: fcct1-3.

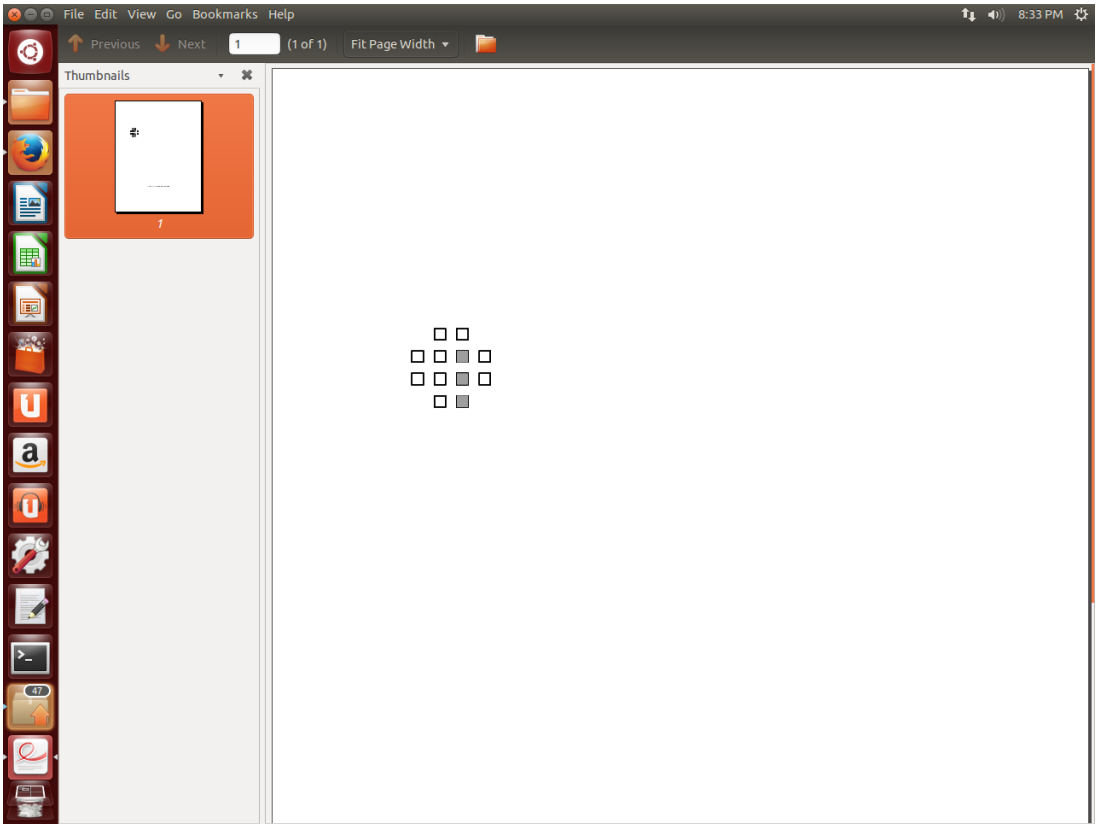

Fig. 5. Circuit: 1-4.

Program Listing
The program code forms the core of the simulated annealing computational program as written in file: ProjectFPGA.cpp and is included in the appendix. In summary is a list of all program subroutines:

List of functions:
Main
SimulatedAnnealing –central program function, takes input from benchmark circuit
InitTemp
Cost
RandomPlacement –pseudo random placement of logic blocks
RangeMove –computes range of block moves
MovesPerTemp –computes ratio of moves given current temperature
Moves -performs standard block movements
DrawFPGA -starts the graphical output of block placement
DrawRoutine
Delay
DeltaCost
Structure LogicBlock
Vector LBlock, Vector costwire, nets

Appendix

```
//Simulated Annealing algorithm Project by: Spiros Hippolyte: EE8219 FPGA Architecture.
#include "stdafx.h"
#include <fstream>
#include <iostream>
#include <stdlib.h>
#include <iomanip>
#include <math.h>
#include <string>
#include <random>
#include "graphics.h"
#include "easygl_constants.h"
#include <vector>
#include <chrono>

using namespace std;

// Callbacks for event-driven window handling.
void delay(void);
void drawscreen(void);
//void act_on_new_button_func(void(*drawscreen_ptr) (void));
//void act_on_button_press(float x, float y);
//void act_on_mouse_move(float x, float y);
//void act_on_key_press(char c);

static bool line_entering_demo = false;
static bool have_entered_line, have_rubber_line;
static bool rubber_band_on = false;
static float xx1, yy1, xx2, yy2;
static int num_new_button_clicks = 0;
//static void close_postscript(void);
```

```
//static int init_postscript(const char *fname);   /* Returns 1 if successful *

class SimulatedAnnealing
{
        private:
                double T; //inital T
                int R, N; //allow long distance moves
                double C; //computed cost
                double a=0, S=0, Snew=0; //alpha
                int nets=0; //# of nets in common
               string circuit;
               struct LogicBlock
               {
                        int x, y, pin, netgroup;  // #net group
               };

        std::vector<int> costwire, common_nets;
                std::vector<LogicBlock> LBlocks;



        public:
                SimulatedAnnealing();                                                   //default
constructor
                SimulatedAnnealing(string circuit, ifstream& file,int& val);            //N logic
clusters,
                //int List(int r, int x, int y, int pin1, int x2, int y2, int pin2, int nets,vector<LogicBlock>
LBlocks, vector<int>& common_nets);   //pin #, x,y location
                void RandPlacement(int r, vector<LogicBlock>& LBlocks, vector<int>& common_nets);
                int Cost(int r, vector<LogicBlock>& LBlocks);                   //semi - perimeter costs
                int MovesPerTemp(int N);
                double InitTemp(int N, vector<int>& costwire, vector<LogicBlock>& LBlocks);
                int RangeMove(double a,int N);
                double TempUpdate(double T, double a, int NetCost,int nets );
                int DeltaCost(vector<int>& costwire, vector<LogicBlock>& LBlocks);
                int Moves(int R, vector<LogicBlock>& LBlocks);
                void delay(void);

                void DrawFPGA(int arraysize, vector<LogicBlock>& LBlocks, int& val);
                void DrawRoutine(int arraysize, vector<LogicBlock>& LBlocks);

};

SimulatedAnnealing::SimulatedAnnealing(void){}

SimulatedAnnealing::SimulatedAnnealing(string circuit, ifstream& file,int& val)
{
        double DC=0,C=0,a=0,rm=0, T=1000;
        int  counter = 0, r = 0, w = 0, x = 0, y = 0, pin = 0, x2 = 0, y2 = 0, pin2 = 0, count = 0;
        char ch;
        ofstream data;
```

```
        while (ch = file.peek() != EOF)  //check next character
        {
                if (count == 0)
                {
                        file >> r;                      //grid array size = [data x data]
                        file >> w;
                        count++;
                }
                else
                {
                        file >> x >> y >> pin >> x2 >> y2 >> pin2;                              //eg. 0 1
0 1 2 3 Pin 0 at (0,1) is connected to pin 3 at (1,2)
                        /*file.getline(line, MAXCHARS, '\n');
                        x = line[0]-48;
                        y = line[2]-48;
                        pin = line[5]-48;*/
                        if (x && y && pin && x2 && y2 && pin2 == -1)
                                file.close();       //end of line for routing file check

                        else
                        {
                                //List(r, x, y, pin, x2, y2, pin2, nets, LBlocks, common_nets);
                                int net_num, LUT = 0;   //to N-1
                                net_num = 1;


                                if (x == x2 && y == y2)
                                {
                                        //same block detected -on same net
                                        LBlocks.push_back({ x, y, pin, net_num });  //push one block
only
                                        nets++; net_num++;
                                }
                                else if (LBlocks.size() != 0)  //check size for empty, not line 0/ 1st
element
                                {
                                        //LUT++;
                                        for (int j = LUT; j < r; j++)
                                        {
                                                //if (LBlocks.at( j).x == LBlocks.at(j+1).x &&
LBlocks.at( j).y == LBlocks.at(j+1).y);//check between rows
                                                if (x == LBlocks.at(j).x && y == LBlocks.at(j).y || x2 ==
LBlocks.at(j).x && y2 == LBlocks.at(j).y)  //check between current and past LUT
                                                        nets++;

                                        }
                                }
                                else
                                {
```

```
                                        LBlocks.push_back({ x, y, pin, net_num });  //x,y,pin,net
number group for unique nets
                                        net_num++; nets++;
                                        LBlocks.push_back({ x2, y2, pin2, net_num });
                                        nets++;

                                }

                                common_nets.push_back(nets);
                        }
                }

        }

        N=r;            //row/col size
        R = N;   //max range of move -inital
        N *=N;                                  //r x r array of N logic block clusters
        int M=0;

        circuit = "Placement_" + circuit+".txt";

        /*unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
        default_random_engine generator(seed);
        normal_distribution<double> Ratio(0.15, 0.8);*/ //Start with best value of alpha --ghamma =0.95
        a= 0.8; //=abs( Ratio(generator)); alpha set to 0.8
        //DrawFPGA(R, LBlocks);
        for (unsigned int b = 0; b < LBlocks.size(); b++)
        {
                RandPlacement(N,LBlocks,common_nets);

        }
        //DrawFPGA(R, LBlocks,val);


        T=InitTemp(N,costwire, LBlocks);
        R=RangeMove(a,R);
        counter++;
        M = MovesPerTemp(N); //M moves

        while (T > 0){
                while (counter < M)
                {
                        RandPlacement(N, LBlocks, common_nets);
        // new placement
                        //DrawFPGA(R, LBlocks);
                        M= Moves(RangeMove(a, R), LBlocks);
                        DC = DeltaCost(costwire,LBlocks);                                       //Cost
of Snew-S=DeltaCost
                        unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
                        default_random_engine generator(seed);                          //r =
randomize(0, 1);  //uniform distrobution
```

```
                        normal_distribution<double> Rand(0.0,1.0);
                        rm = abs(Rand(generator));
                        if (rm < exp(-(DC / T)))
                        {
                                RandPlacement(N, LBlocks, common_nets);
                //update S 'new
                                //DrawFPGA(R, LBlocks);
                        }


                        counter++;

                }
                C = Cost(r, LBlocks);
                T = TempUpdate(T, a, C,nets );
                R = RangeMove(a, R);                                                    //update


                data.open(circuit, ios::app);                                           //send to file
                if (data.fail())
                {
                        cout << "The file failed to be opened" << endl;
                        system("pause");
                        exit(1);
                }

                cout << "The semi-perimeter minimum cost is: " << C << " given " << M << " moves per
configuration" << endl;  //send to screen
                data << "The semi-perimeter minimum cost is: " << C << " given " << M  << " moves
per configuration" << endl;  //send to file
                costwire.clear(); common_nets.clear();LBlocks.clear();
                data.close();
        }
        data.open(circuit, ios::app);
        if (data.fail())
        {
                cout << "The file failed to be opened" << endl;
                system("pause");
                exit(1);
        }
        cout << "The semi-perimeter minimum cost is: " << C << " given " <<  M << " moves" << endl;
        data << "The semi-perimeter minimum cost is: " << C << " given " <<  M << " moves" << endl;
//send to file
        data.close();

        DrawFPGA(R, LBlocks,val);

}

void SimulatedAnnealing::DrawFPGA(int arraysize, vector<LogicBlock>& LBlocks, int& val)
{
```

```
	init_graphics("FPGA placement", WHITE);
	init_world(0., 0., 1000., 1000.);
	update_message("FPGA LUT block randomized ");
	//event_loop(act_on_button_press, NULL, NULL, drawscreen);
	// Enable mouse movement (not just button presses) and key board input.
	// The appropriate callbacks will be called by event_loop.
	set_keypress_input(true);
	set_mouse_move_input(true);
	//line_entering_demo = true;

	char fileO[30] = "Circuit";


	init_postscript(fileO);
	DrawRoutine( R,LBlocks);
	//event_loop(act_on_button_press, NULL, NULL, drawscreen);
	/*flushinput();
	delay()*/;

	close_postscript();
}

void SimulatedAnnealing::DrawRoutine(int arraysize, vector<LogicBlock>& LBlocks)
{
	/*t_point polypts[3] = { { 500., 400. }, { 450., 480. }, { 550., 480. } };
	t_point polypts2[4] = { { 700., 400. }, { 650., 480. }, { 750., 480. }, { 800., 400. } };*/
	set_draw_mode(DRAW_NORMAL);  // Should set this if your program does any XOR drawing
in callbacks.

	clearscreen();
	//drawrect(100, 100, 900, 900);
	setfontsize(10);
	setlinestyle(SOLID);
	setlinewidth(1);
	setcolor(BLACK);


	for (int q=0; q < arraysize; q++)
	{

		for (int i = 0; i<(arraysize + 2); i++){
			for (int j = 0; j<(arraysize + 2); j++){
				if ((i == 0 && j == 0) || (i == 0 && j == (arraysize + 1)) || (j == 0 && i
== (arraysize + 1)) || (i == (arraysize + 1) && j == (arraysize + 1)))
					;
				else {
					setcolor(BLACK);
					drawrect((j*40. + 20.), (i*40. + 20.), (j*40. + 40.), (i*40. + 40.));
				}
			}
		}
```

```
		int x = 0, y = 0;
		for (unsigned int i = 0; i<LBlocks.size(); i++)		//given size of array and current blocks
		{
			y = LBlocks.at(i).y;
			x = LBlocks.at(i).x;
			setcolor(DARKGREY);
			fillrect((y*40. + 20.), (x*40. + 20.), (y*40. + 40.), (x*40. + 40.));
		}

		have_rubber_line = false;		// Screen redraw will get rid of a rubber line.

	}

	/*int init_postscript(string Output);


	void close_postscript(void);*/
}

//int SimulatedAnnealing::List(int r, int x, int y,int pin1, int x2, int y2, int pin2,int nets,vector<LogicBlock> LBlocks,vector<int>& common_nets)
//{
//	int net_num, LUT=0;	//to N-1
//	net_num = 1;
//
//
//		if (x == x2 && y == y2)
//		{
//			//same block detected -on same net
//			LBlocks.push_back({ x, y, pin1, net_num });  //push one block only
//			nets++; net_num++;
//		}
//		else if (LBlocks.size() !=0)  //check size for empty, not line 0/ 1st element
//		{
//			//LUT++;
//			for (int j = LUT; j < r ;j++)
//			{
//				//if (LBlocks.at( j).x == LBlocks.at(j+1).x && LBlocks.at( j).y == LBlocks.at(j+1).y);//check between rows
//				if (x == LBlocks.at(j).x && y == LBlocks.at(j).y  || x2 == LBlocks.at(j ).x && y2 == LBlocks.at(j ).y)  //check between current and past LUT
//					nets++;
//
//			}
//		}
//		else
//		{
//			LBlocks.push_back({ x, y, pin1,net_num });  //x,y,pin,net number group for unique nets
```

```
//                          net_num++; nets++;
//                          LBlocks.push_back({ x2, y2, pin2, net_num });
//                          nets++;
//
//                  }
//
//          common_nets.push_back(nets);
//          return nets;
//}

 void SimulatedAnnealing::RandPlacement(int N, vector<LogicBlock>& LBlocks, vector<int>&
common_nets)
{
          //randomly place N logic clusters /random logic location found given array size
           int  x,y;
          //locations (0,0) to (x+r, y+r) of N logic clusters
          unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
          std::default_random_engine generator(seed);
          //default_random_engine generator;
          std::uniform_int_distribution<int> distribution(1,N);
          //random number generator(0 to r)



                  for (unsigned int i=0; i < LBlocks.size(); i++)
                          {
                                  x = distribution(generator);
                                  y = distribution(generator);

                                  if (i == 0)
                                  {

                                          LBlocks.at(i).x = x;
                                          LBlocks.at(i).y = y;
                                          nets++;//new random co-ordinates for blocks
                                          cout << LBlocks.at(i).x << " " << LBlocks.at(i).y << endl;
                                  }

                                  else
                                  {
                                          for (unsigned int m = 0; m < i;m++)
                                          {
                                                  if (LBlocks.at(i).x == LBlocks.at(m).x &&
LBlocks.at(i).y == LBlocks.at(m).y)      //not unique position
                                                  {
                                                          cout << " Block present" << endl;
                                                          x = distribution(generator);
                                                          y = distribution(generator);
                                                          LBlocks.at(i).x = x;
                                                          LBlocks.at(i).y = y;                  //new random
co-ordinates for blocks
```

```
                                                cout << LBlocks.at(i).x << " " <<
LBlocks.at(i).y << endl;
                                                nets++;
                                        }
                                        else
                                        {
                                                //LBlocks.push_back({ x, y });

                                                LBlocks.at(i).x = x;
                                                LBlocks.at(i).y = y;
                                                nets++;
                                                cout << LBlocks.at(i).x << " " <<
LBlocks.at(i).y << endl;
                                        }
                                }
                        }
                }
        common_nets.push_back(nets);

}

 int SimulatedAnnealing::Cost(int r, vector<LogicBlock>& LBlocks)
{
        //cost of wire length
        int  wirelength=0, A = 0,B = 0, minx=r,miny=r, maxx=0,maxy=0,tempx=0,tempy=0;  //semi -
perimeter ;



        for (unsigned int i = 0; i < LBlocks.size(); i++)               //compute costs //per net # groups
        {
                //cost/net
                if (LBlocks.at(i).x < minx)
                        minx = LBlocks.at(i).x;
                        if (LBlocks.at(i).y = miny)
                                miny = LBlocks.at(i).y;
                if (LBlocks.at(i).x > maxx)
                        maxx = LBlocks.at(i).x;
                if (LBlocks.at(i).y>maxy)
                        maxy = LBlocks.at(i).y;

        }
                A = abs(maxx-minx);  //cost A
                B = abs(maxy-miny);   //cost B
                wirelength = A + B;                                     //semi -perimeter length of wire
length cost
                costwire.push_back({ wirelength });             //add to semi-perimeter vector set

        return wirelength;
}
```

```
int SimulatedAnnealing::MovesPerTemp(int N)
{
	double C = 1.0;
	int MoveperT;
		MoveperT = pow(C*N, (4 / 3));  //C*N^ (4/3)
		MoveperT = round(MoveperT);
	return MoveperT;
}

int SimulatedAnnealing::RangeMove(double a, int R)
{
	R= R*(1 - 0.44 + a);   //moves are integer
	R=round(R);                              //round to nearest interger
	return R;
}

int SimulatedAnnealing::Moves(int R, vector<LogicBlock>& LBlocks)
{
	int x, y,m=0;                                                    //location data in vector
                                                                                //random placement initally
performed
	for (int k = 0; k < N; k++)                                  //N array logic block moves
	{
		x = LBlocks.at(k).x;                                  //call back set 0 x,y then set 1: x,y
		y = LBlocks.at(k).y;
		x += R;                                                          //logic block move given R
range
		y += R;
		LBlocks.push_back({ x, y });
	}
	m++;
	return m;
}

double SimulatedAnnealing::InitTemp(int N, vector<int>& costwire, vector<LogicBlock>& LBlocks)
{
	double stddev=0,var=0,mean=0;
	int sum=0;                                  //N logic blocks


	//N = LBlocks.size();

	for (int z = 0; z < N; z++)                          //N move for N configurations
	{
		Cost(R, LBlocks);
		sum += costwire.at(z);
	}
	mean = sum / N;                                                  //mean calculation

	for (int j = 0; j < N;j++)
```

```
	var = pow( (costwire.at(j)-mean ), 2);			//varians of set of cost per
configurations

	stddev = sqrt(var);							//compute
STD.Dev of cost (set) of N configs
	T = 20.0 * stddev;
	return T;
}

double SimulatedAnnealing::TempUpdate(double T, double a, int NetCost,int nets)
{
	double Gamma=1;
	double e= 0.005;  //Epsilion %error
	if (a > 0.96)
		Gamma = 0.5;
	else if (a >= 0.8 && a <= 0.96)
		Gamma = 0.9;
	else if (a >= 0.15 && a <= 0.8)
		Gamma = 0.95;  //best move
	else if (a < 0.5)
		Gamma = 0.8; //best cost


	if (T < e * NetCost/nets);			//cost per nets ->
			T *= Gamma;


		return T;
}

int SimulatedAnnealing::DeltaCost(vector<int>& costwire, vector<LogicBlock>& LBlocks)
{
	//cost A-B
	int result=0,A=0,B=0;
	//cost of A

	B = costwire.at(N-1);  //old previously computed
	RandPlacement(N,LBlocks,common_nets);	//Snew cost set
	A = Cost(R, LBlocks);
	//new
	if (A < B)			//A is optimum -local
	{
		result = B - A;
		costwire.clear();
		LBlocks.clear();
	}

	else if (B < A)
	{
		result = A - B;//cost of new set of random placement subtract, computed cost of past set
of random placements
```

```
			costwire.clear();
			LBlocks.clear();
		}

		return result;
}

int _tmain(int argc, _TCHAR* argv[])
{
		const int MAXLENGTH = 10;
		const int MAXCHARS = 20;
		ifstream file;
		//char CircuitN[MAXCHARS];
		//char ch;
		char file1[MAXLENGTH] = "fcct1";
		char file2[MAXLENGTH] = "fcct1-1";
		char file3[MAXLENGTH] = "fcct1-2";
		char file4[MAXLENGTH] = "fcct1-3";
		char file5[MAXLENGTH] = "fcct1-4";
		char file6[MAXLENGTH] = "fcct2";
		char file7[MAXLENGTH] = "fcct2-4";
		char file8[MAXLENGTH] = "fcct2-5";
		char file9[MAXLENGTH] = "fcct3";
		char file10[MAXLENGTH] = "fcct3-8";
		char file11[MAXLENGTH] = "fcct4";
		char file12[MAXLENGTH] = "fcct4-14";
		string FileSel;
		//ofstream outfile;


		//void DrawG();
		int val=0;
		SimulatedAnnealing SA;
		cout << "Please select one of the benchmark circuit files" << endl;
		cout << "Select the benchmark circuit files from 1 to 12 by typing in the number" << endl;
		cout << " fcct1    (1)\n fcct1 - 1 (2)\n fcct1 - 2 (3)\n fcct1 - 3 (4)\n fcct1 - 4 (5)\n fcct2     (6)\n
fcct2 - 4 (7)\n fcct2 - 5 (8)\n fcct3     (9)\n fcct3 - 8 (10)\n fcct4     (11)\n fcct4 - 14 (12)" << endl;

		cin >> val;
			switch (val)
			{
			case 1:
				file.open( file1, ios::in); FileSel = file1;
				break;
			case 2:
				file.open(file2, ios::in);  FileSel = file2;
				break;
			case 3:
				file.open(file3, ios::in);  FileSel = file3;
				break;
			case 4:
```

```
                        file.open(file4, ios::in);  FileSel = file4;
                        break;
                case 5:
                        file.open(file5, ios::in);  FileSel = file5;
                        break;
                case 6:
                        file.open(file6, ios::in);  FileSel = file6;
                        break;
                case 7:
                        file.open(file7, ios::in);  FileSel = file7;
                        break;
                case 8:
                        file.open(file8, ios::in);  FileSel = file8;
                        break;
                case 9:
                        file.open(file9, ios::in);  FileSel = file9;
                        break;
                case 10:
                        file.open(file10, ios::in);  FileSel = file10;
                        break;
                case 11:
                        file.open(file11, ios::in);  FileSel = file11;
                        break;
                case 12:
                        file.open(file12, ios::in);  FileSel = file12;
                        break;
                }

        if (file.fail())
        {
                cout << "\nThe file failed to open, check file location exisitance" << endl;
                system("pause");
                exit(1);
        }

        SimulatedAnnealing ASmain(FileSel, file,val );          //Call main computation program,
forward input benchmark circuit file name
        //DrawG();



        system("pause");
        close_graphics();
        return 0;

}

void delay(void) {

        /* A simple delay routine for animation. */
```

```
        int i, j, k, sum;

        sum = 0;
        for (i = 0; i<100; i++)
                for (j = 0; j<i; j++)
                        for (k = 0; k<1000; k++)
                                sum = sum + i + j - k;
}
```