CCirc and CGen User's Manual

Version 1.0

Paul D. Kundarewich (paul.kundarewich@utoronto.ca)

Jonathan Rose, Mike Hutton

http://www.eecg.toronto.edu/~jayar/software/Cgen/Cgen.html

February 1, 2003

1 Overview

The development of next-generation CAD tools and FPGA architectures require benchmark circuits to experiment with new algorithms and architectures. There has always been a shortage of good public benchmarks for these purposes, and even companies that have access to proprietary customer designs could benefit from designs that meet size and other particular specifications. This document describes two new tools CCirc and CGen to help alleviate this shortage. These tools significantly improves the quality of previous work by imposing the natural hierarchy of circuits through clustering and by using a simpler method of characterizing the nature of sequential circuits. Also, in contrast to current constructive generation methods, we employ new iterative techniques in the generation that provide better control over the generated circuit's characteristics.

CCirc characterize physical properties of circuits. It takes as input a circuit in the BLIF netlist format (soon VHDL) and outputs statistical information.

CGen generates new synthetic circuits. It takes as input statistical information about a circuit and outputs netlists in BLIF or VHDL ready to be place and routed.

Executables are available for Linux, Solaris, and Windows and source code is provided for those who want to modify the code or for those who want to compile for other platforms.

The user's guide is organized as follows. Section 2 describes how to run CCirc and CGen to analyze circuits and to generate synthetic circuits. Section 3 describes how to compile CCirc and CGen. Section 4 discusses very briefly the characteristics of a circuit that form the basis of the output of CCirc and for the input to CGen. More information on the characterizations can be found from the documents in the biography. Section 5 illustrates a simple example of using CCirc and CGen together to generate a synthetic circuit. Section 6 will (eventually) discuss more advanced usage of CGen such as the problems involved in modifying the characterizations from CCirc to generate different and/or larger circuits. Section 7 states what is currently not supported.

2 Using the Executables

This section describes the operation of CCirc and CGen.

2.1 Operation of CCirc

CCirc takes as input a technology-mapped netlist of lookup tables (LUTs) and flip flops in BLIF format, and outputs a `.stats` file containing the result of circuit characterization. A brief description of these characterizations can be found in Section 4. CCirc is invoked by typing:

```
ccirc netlist.blif [Options…]
```

In the command above, `netlist.blif` should be replaced by the BLIF file you are analyzing.

Options for CCirc are listed in Table 2.1. The most important options is **`--partitions`** as this controls the number of clusters created in the characterization. In general the default values for the rest of the options are fine.

Table 2.1 CCirc Options

Option	**Default**	**Description**
--out <file_name>	netlist.stats where "netlist" comes from the input file's name.	The output file containing the characterizations of the circuit.
--partition type bi|kway	Kway	The type of partitioning.
--partitions <int>	2	The number of clusters to divide the circuit into.
--wirelength approx	Don't calculate	Calculate the wirelength$_{approx}$.
--draw	Don't draw	Outputs a drawing of the circuit in dot format.
--help		Displays list of options.
--no warn	Warn the user	Suppresses warning messages.

2.2 Operation of CGen

CGen takes as input a `.stats` file (possibly modified) from CCirc and outputs a circuit in the BLIF format where all gates in the circuit have been mapped to NAND gates. CGen is invoked by typing:

```
cgen netlist.stats [Options…]
```

In the command above, `netlist.stats` should be replaced by the `.stats` file you want to generate from.

Options for CGen are summarized in Table 2.2 and are described in the sections below. In general, the default values of the options are fine and only people looking to explore the algorithms used in CGen will use many of them. To output the synthetic circuit in the VHDL format please use the option **–-vhdl_output**.

Table 2.2 CGen Options Summary

Option	Default
--out output_file	netlist_clone.blif
--vhdl_output	
--seed <int>	Clock ticks from system clock
--draw	
--help	
--alpha <float>	1
--beta <float>	1
--gamma <float>	1
--delay_structure_init_temperature	6
--init_too_many_inputs_factor <float>	1
--init_too_few_inputs_factor <float>	1
--init_too_few_outputs_factor <float>	1
--mult_too_many_inputs_factor <float>	1.5
--mult_too_few_inputs_factor <float>	1.5
--mult_too_few_outputs_factor <float>	1.5
--delay_structure_init_temperature <float>	1
--delta <float>	0.5
--degree_init_temperature <float>	1
--eta <float>	0.02
--wirelength {init | stats | <int> }	Combinational Circuits: init Sequential Circuits: 0
--final_edge_assignment_dff_loop_cost <float>	10
--final_edge_assignment_init_temperature <float>	0.0000001

2.2.1 General Options

--draw Outputs a drawing of the synthetic circuit in dot format.

--no_warn Suppresses warning messages to the user.

--seed <int> The random number generator seed. The default value is to randomly seed the generator with the number of clock times from the system clock.

--vhdl_output Outputs the synthetic circuit as a VHDL file instead of a BLIF.

2.2.2 Delay Structure Creation Options

In delay structure creation we are trying to minimize the cost:

$$cost_{Delay\ structure} = \alpha\ \mathrm{cost}_{Comb} + \beta\, cost_{Edge\ Length} + \gamma\,(cost_{Level\ Shape} + cost_{Problem\ Node})$$

Here, $cost_{Comb}$ measures the absolute difference between the current *Comb* and its specification, $cost_{EdgeLength}$ measures the absolute difference between the current edge length distributions and their specifications, $cost_{LevelShape}$ measures the absolute difference between the current input and output shapes and their specifications with congestion factors multiplying this cost at each level node to penalize level nodes that have too many inputs or outputs. Finally, $cost_{Problem\ Node}$ measures the number of node violations that would be forced to be made during final edge assignment because of the number of edges input or output of a level node. In the cost function alpha, beta, and gamma are factors that try to balance the individual costs against each other although the cost that tends to dominate is the $cost_{Problem\ Node}$ because the congestion factors tend to get quite large.

--alpha <float> Sets the multiplier of $cost_{Comb.}$ Default 1.

--beta <float> Sets the multiplier of $cost_{EdgeLength}$. Default 1.

--gamma <float> Sets the multiplier of $cost_{Level\_Node.}$ Default 1.

--init_too_many_inputs_factor <float> Sets the starting value of the congestion factor that penalizes level nodes that have more input edges than k * number of indiv. nodes. Default 1.

--init_too_few_inputs_factor <float> Sets the starting value that penalizes level nodes that do not have enough input edges for all of their indiv. nodes. Default 1.

--init_too_few_outputs_factor <float> The initial congestion factor that penalizes level nodes that do not have enough output edges for all of their indiv. nodes. Default 1.

--mult_too_many_inputs_factor <float> Multiplies too_many_inputs_factor by this amount when increasing the congestion costs. Default 1.5.

--mult_too_few_inputs_factor <float> Sets the growth factor that multiplies too_few_inputs_factor when increasing the congestion factors. Default 1.5.

--mult_too_few_outputs_factor <float> Sets the growth factor that multiplies too_few_outputs_factor when increasing the congestion factors. Default 1.5.

--delay_structure_init_temperature <float> Sets the initial temperature of the annealing schedule. Default 6.

2.2.3 Degree Partitioning Options

In final edge assignment we are trying to minimize the cost:

$$cost_{Degree} = delta \sum_{LN \in Level\ Nodes} \text{cost}_{Fanout\ Edge\ Misassignment}(LN) + (1 - delta)\,\text{cost}_{Fanout\ Penalty}(LN)$$

Here $cost_{FanoutEdgeMissassignment}$ measures for each level node the absolute difference between the sum of the fanout degrees assigned and the number of output edges that were assigned during the creation of the delay structure and $cost_{FanoutPenality}$ is a cost that penalizes level nodes with fanouts that will force node violations in the final edge assignment. We use the factor *delta* to balance the goal of achieving the a match between fanout and edge assignment against the goal of having no node violations.

--delta <float> Degree Partitioning cost trade-off. Default 0.5.

--degree_init_temperature <float> Sets the temperature of the anneal. Default 1.

2.2.4 Final Edge Assignment Options

In final edge assignment we are trying to control the wirelength and we are trying to prevent node violations. The cost function for circuit G is:

$$cost_{Edge\ Assign} = eta(\text{wirelength}_{\text{Approx}}(\text{G}) - desired\_wirelength) + (1 - eta)\sum_{n \in V(G)} \text{Number of Violations}(n)$$

Here, *desired wirelength* is the desired $wirelength_{Approx}$ and *Number of Violations* is a function that returns the number of nodes that have no inputs, too many inputs, two or more connections from the same source node, or if the node is a flip-flop a connection to itself. The wirelength costs are normalized to the maximum horizontal position multiplied by the number of edges while the *Number of Violations* cost is normalized to the number of edges. We use the factor eta to balance the goal of achieving the desired wirelength against the goal of having no node violations. The value is set to 0.02 because the wirelength cost is often much larger than the no node violation cost and while achieving the desired wirelength is important it is more important that we have no node violations because they create sizeable difficulties.

--eta <float> Final edge assignment cost trade-off. Default 0.02.

--wirelength { init | stats | <int> } Sets the desired_wirelength $Wirelength_{approx}$ To use the $wirelength_{Approx}$ of the initial solution choose **init**. To use the $wirelength_{Approx}$ from the stats choose **stats.** To use your own value of $wirelength_{Approx}$ enter an integer. The default for combinational circuits is **init**. The default value for sequential circuits is the integer 0 to force solutions to seek minimum wirelength.

--final_edge_assignment_dff_loop_cost <float> Sets the factor by which to extra penalize the node violations where flip-flops that connect to themselves. Default 10.

--final_edge_assignment_init_temperature <float> Sets the temperature of the anneal. Default `0.0000001.`

3 Compiling CCirc and CGen

This section describes how to compile CCirc and CGen.

3.1.1 Compiling CCirc

This section describes how to compile CCirc. All versions of CCirc require the hMetis library partitioner which is available at http://www-users.cs.umn.edu/~karypis/metis/hmetis/.

In the source code I have included files flex and bison generated code if you do not possess bison or flex (although versions exists on the web for most platforms including Windows).

3.1.2 Compiling Under Linux or Solaris

If your are compiling under Linux or Solaris the compilation process is simple and has four steps:

Step 1: Make sure your version of `gcc` is at least 2.95.

Step 2: Download the hMetis library and save it somewhere.

Step 2: In the file `Makefile` set `PARTITION` to point to where the hMetis library is located.

Step 4: type `make`.

Step 5: Look at your new executable `ccirc.`

3.1.3 Compiling Under Windows

If your are compiling under Windows the compilation process is little more complicated. I have used STL hash_maps which is not included by default by Visual C++ 6.0[1]. To solve the missing hash_map class I used the STLport located at http://www.stlport.org/. CCirc also uses the hMetis partitioner and its use under windows also requires a little work.

The compilation of CCirc under Windows has 5 steps:

Step 1: Download the hMetis library

Step 2: Convert the .obj files stored in libhmetis.a into an libhmetis.lib.

2.1 libhmetis.a is as far as I can tell a UNIX archive of Windows object files.

One way to extract them is by in UNIX/Linux running: ar x libhmetis.a and then transferring the obj files to a Windows machine.

2.2. Once they have been extracted and transferred a .lib can be built by:

Running: lib.exe file1.obj file2.obj ... /OUT:libhmetis.lib

Step 3: Download and install STLport without STLports iostreams as instructed in the STLport documentation.

[1] hash_map is available under Visual C++ 7.0 but I have not tried to compile under version 7.0.

Step 4: Start Visual C++ and open the workspace Cgen/ccirc/ccirc/ccirc.dsw

Step 5: Under *Project->Settings* Click on the *Link* Tab

Step 6: Under the *Link* tab change the *Category: General* to *Category: Input.*

Step 7: In the Additional library path change the location of the hmetis.lib if incorrect. Then Click *ok*.

Step 8: Under Build choose Batch build.

Step 9: Look at your new executable `ccirc`.

3.2 Compiling CGen

Compiling CGen is much simpler than compiling CCirc as no external libraries are required and the code does not use flex/bison.

3.2.1 Compiling Under Linux or Solaris

If your are compiling under Linux or Solaris the compilation process is simple and has four steps:

Step 1: Make sure your version of `gcc` is at least 2.95.

Step 2: Change to the cgen directory

Step 3: type `make`.

Step 4: Look at your new executable `cgen`.

3.2.2 Compiling Under Windows

If your are compiling under Windows the compilation process is also relatively simple:

Step 1: Start Visual C++ and open the workspace Cgen/cgen/cgen.dsw

Step 2: Under Build choose Batch build.

Step 3: Look at your new executable `cgen`.

4 Circuit Characterizations

In this section we very briefly discusses the characteristics of a circuit that form the basis of the output of CCirc and for the input to CGen. More information on the characterizations can be found in my thesis. A sample of the statistical information output from CCirc can be seen in Figure 1in Section 5.

4.1 Circuit Model and Definitions

Circuits are modeled as a directed acyclic graph $G = (V,E)$ where the nodes V represent gates in the circuit and edges E represent two-point connections between gates.

4.2 Basic Characterizations of Circuit

The most basic parameters of a circuit are the following:

Circuit Name

The name of the circuit

Number of Nodes

The number of nodes in the circuit or cluster. A node is a primary input, a LUT/logic gate, or a flip-flop.

Number of Edges

The number of edges in the circuit or cluster.

Maximum Delay

The maximum combinational delay in the circuit.

Number of PI

The number of primary inputs in the circuit.

Number of PO

The number of primary outputs in the circuit.

Number of Combinational Nodes

The number of LUTs/logic gates in the circuit.

Number of DFF

The number of flip-flops in the circuit

kin

The lut-size/maximum gate fanin in the design.

clock

The clock name if it is a sequential circuit.

Number of Clusters

The number of clusters the circuit has been partitioned into.

4.3 Characterization of Delay Structure

A key concern of modern digital design is the speed at which circuits operate and therefore we employ the unit delay model in which every gate incurred a single unit of delay. With this delay model, the *delay level* of a node in the graph is defined as the maximum delay over all directed paths beginning at a primary input (PI) or a flip-flop (DFF) and terminating at the given node.

The delay structure of the circuit is characterized by a collection of measurements at the various delay levels. *Shape* is defined as the number of objects at each delay level. Accordingly, we define:

Node Shape

A measurement of shape for the number of nodes at each delay level

Input Shape

A measurement of shape for the number of inputs into each delay level

Output Shape

A measurement of shape for the number of inputs into each delay level

POShape

A measurement of shape for the number of primary outputs at each delay level

Latched_Shape

A measurement of shape for the number of nodes whose output drives the input of a flip-flops or are "latched" at each delay level.

4.4 Characterization of Connections

To characterize the connections in the combinational circuit, we define an edge length property: For an edge e=(x,y) with nodes x and y we define the length(e) = delay_level(y) - delay_level(x) if delay_level(y) > delay_level(x). An edge of length 1 is termed a unit edge while any edge with a length greater than 1 is termed a long edge.

With edge length, we can define an edge length distribution for the circuit which is the number of edges at each edge length.

Inside each cluster we have edges that are internal to the cluster, that input into the cluster, and that output from the cluster. As such we define an edge length distribution for each type of edge.

Intra-cluster Edge Length Distribution

The number of edges internal to a cluster at each edge length

Inter-cluster Input Edge Length Distribution

The number of inter-cluster inputs into a cluster at each edge length

Inter-cluster Output Edge Length Distribution

The number of inter-cluster outputs out of a cluster at each edge length

4.5 Characterization of Fanout from Nodes

For a node *x*, *fanout(x)* is the number of connections to combinational nodes. For a circuit we describe the fanout in terms of the *fanout distribution,* defined as the number of nodes of each fanout, starting at 0.

Fanout Distribution

The number of nodes of each fanout, starting at 0.

Maximum_fanout

The maximum fanout of a node in the circuit.

We also have a number of other fanout statistics which are output by CCirc related to the fanout and fanin of the primary inputs, combinational nodes (LUTs), and the flip-flops in the circuit. Their values are not as important as the other statistics in the file and with the exception of Avg_fanout_pi and Avg_fanout_dff are not use in synthetic circuit generation at present.

4.6 Characterization of Wirelength

To control the post-place and route wirelength of the synthetic circuits that we output from our generation process we measure an approximation to wirelength in characterization that can be used in generation.

Wirelength_approx

An approximation to wirelength in the circuit.

4.7 Characterization of the Inter-cluster Connectivity

The structure of the connections between clusters is then captured through two matrices that count the number of connections to combinational nodes and flip-flops between clusters. The first matrix we define as Comb=[combij] where combij is the number of inter-cluster connections that drive combinational nodes from clusters Ci to Cj. The second matrix we define as Latched=[latchij] where latchij is the number of connections that drive flip-flops from Ci to Cj.

Comb

A matrix that measures the number inter-cluster connections to combinational nodes

Latched

A matrix that measures the number latched node to flip-flop connections between clusters

5 Using CCirc and CGen

As a simple illustration of using together CCirc and CGen we will generate a synthetic circuit from the MCNC circuit *diffeq.blif*.

Step 1: Analyze the diffeq circuit by running CCirc.

```
ccirc diffeq.blif --partitions 4
```

This will produce a *diffeq.stats* file similar to the .stats file below in Figure 1.

Step 2: Modify the characterizations in *diffeq.stats* if a synthetic circuit with different properties is desired.

Step 3: Generate a synthetic circuit by running CGen on *diffeq.stats*.

a) To produce a synthetic circuit in BLIF format:

```
cgen diffeq.stats
```

b) To produce a synthetic circuit in VHDL format:

```
cgen diffeq.stats –vhdl_output
```

Figure 1 - A sample of the statistical output from CCirc

Note: The text in *italics* in this .stats file are comments used to illustrate a .stats file and do not exist in a real .stats file. The statistics mentioned are the same as those is Section 4.

```
######################## BASIC ############################
```

Basic statistics about the circuit.

```
Circuit_Name:  diffeq
Number_of_Nodes:  1934
Number_of_Edges: 5631
Maximum_Delay: 14
Number_of_PI: 63
Number_of_PO: 39
Number_of_Combinational_Nodes: 1494
Number_of_DFF: 377
kin: 4
clock: pclk
======================== Cluster_Summary ==================
```

A summary of the basic statistics divided into the number of elements in each cluster such as, for example, the number of nodes and in each cluster.

```
Number_of_nodes: ( 537 490 457 450 )
Number_of_pi: ( 23 40 0 0 )
Number_of_dff: ( 103 117 87 70 )
Number_of_intra_cluster_edges: ( 1189 1014 1092 1052 )
Number_of_inter_cluster_edges: ( 427 599 379 409 )
```

The Partitioned_scaled_cost is a number that represents the quality of the partitioning performed on the circuit.

```
Partitioned_scaled_cost: 1.2698
======================== DEGREE ============================
```

Statistics related to the fanout and fanin to the primary inputs (pi), combinational (comb), the flip-flops (dff) in the circuit. Their values are not as important as the other statistics and with the exception of Avg_fanout_pi and Avg_fanout_dff are not use in synthetic circuit generation at present.

```
Avg_fanin_comb: 3.51673 (0.662174)
Avg_fanout: 2.91158 (14.2713)
Avg_fanout_comb: 1.7008 (3.84116)
Avg_fanout_pi: 8.92063 (61.8604)
Avg_fanout_dff: 6.70557 (17.9062)
Maximum_fanout: 496
Number_of_high_degree_comb: 12
Number_of_high_degree_pi: 1
Number_of_high_degree_dff: 13
Number_of_10plus_degree_comb: 27
Number_of_10plus_degree_pi: 1
Number_of_10plus_degree_dff: 33
======================= SHAPE ============================
```

Shape and fanout distribution statistics about the circuit as a whole.

```
Node_shape: ( 440 718 277 119 38 46 112 61 39 32 19 15 9 5 4 )
Input_shape: ( 0 2476 895 466 140 175 433 229 137 111 71 55 33 20 13 )
Output_shape: ( 3090 1122 332 150 95 157 112 87 39 24 20 13 8 5 0 )
Latched_shape: ( 0 53 133 39 4 5 81 9 11 20 7 7 3 1 4 )
POshape: ( 3 36 0 0 0 0 0 0 0 0 0 0 0 0 0 )
Edge_length_distribution: ( 0 3813 568 170 98 218 162 90 47 26 27 17 12 4 2 )
Fanout_distribution: ( 374 954 286 94 49 33 33 31 12 7 5 6 15 9 0 0 0 0 0 0 0
0 0 0 4 0 0 0 1 0 0 1 1 0 0 0 2 0 0 0 0 1 1 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 3 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 )
#################### Clusters #####################
```

The start of the statistics broken down cluster by cluster.

```
Number_of_Clusters: 4
#################### Cluster 0 #####################
```

The basic statistics for the first cluster.

```
Number_of_Nodes: 537
Number_of_Intra_cluster_edges: 1189
Number_of_Inter_cluster_edges: 427
Number_of_PI: 23
Number_of_PO: 1
Number_of_Comb: 411
Number_of_DFF: 103
Number_of_Latched: 96
Number_of_Inter_cluster_input_edges: 248
Number_of_Inter_cluster_output_edges: 179
======================= DEGREE ===========================
```

Statistics related to the fanout and fanin to the primary inputs (pi), combinational (comb), the flip-flops (dff) of the cluster.

```
Avg_fanin_comb: 3.49635 (0.732865)
Avg_fanout: 2.72626 (7.89913)
Avg_fanout_comb: 1.74939 (4.59058)
Avg_fanout_pi: 1.17391 (0.636032)
Avg_fanout_dff: 6.97087 (14.7907)
Maximum_fanout: 124
Number_of_high_degree_comb: 4
Number_of_high_degree_pi: 0
Number_of_high_degree_dff: 4
Number_of_10plus_degree_comb: 4
Number_of_10plus_degree_pi: 0
Number_of_10plus_degree_dff: 4
======================= SHAPE ===========================
```

Shape and fanout distribution statistics about the cluster.

```
Node_shape: ( 126 194 119 41 8 12 6 9 9 5 5 3 0 0 0 )
Input_shape: ( 0 670 383 163 32 46 22 35 35 20 19 12 0 0 0 )
```

```
Output_shape: ( 745 344 123 46 30 20 12 19 10 9 6 4 0 0 0 )
Latched_shape: ( 0 4 59 23 1 1 2 1 2 1 2 0 0 0 0 )
POshape: ( 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 )
Intra_cluster_edge_length_distribution: ( 0 948 125 41 21 12 9 19 4 2 4 4 0 0
0 )
Inter_cluster_input_edge_length_distribution: ( 0 204 32 1 1 1 2 1 3 1 2 0 0
0 0 )
Inter_cluster_output_edge_length_distribution: ( 0 138 26 12 0 1 0 1 0 0 1 0
0 0 0 )
Fanout_distribution: ( 95 278 58 16 19 17 14 19 9 4 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 )
################### Cluster 1 #####################
```
The basic statistics for the second cluster.

```
Number_of_Nodes: 490
Number_of_Intra_cluster_edges: 1014
Number_of_Inter_cluster_edges: 599
Number_of_PI: 40
Number_of_PO: 38
```

{cut}

```
-------------------- Inter_cluster_adjacentcy_matrix_to_combinational_nodes -
-------------------
```
The Comb matrix

```
0 79 58 42
241 0 110 150
7 3 0 113
0 16 88 0
```

```
-------------------- Inter_cluster_adjacentcy_matrix_to_dffs ---------------
-------------------
```
The Latched matrix

```
95 1 0 0
0 115 1 0
8 1 84 0
0 0 2 70
```

6 Advanced Generation

Future Research!

7 Not Supported

1. Invisible nodes. Sometimes, especially when building a clock splitter or similar structures, it is possible to have a set of registers and logic which is self-contained and is fed purely from itself (no PIs affect the output) and will just outputs values. This is different from being unreachable from the PIs, because the nodes are affected by the clock. We refer to these nodes as “invisible” because they are not seen from the PIs. We should deal with them but for now we just delete such nodes.
2. Circuits with more than one clock.