
EC440 VLSI Design Automation

Project Documentation

Release 1.0

Rahul M Hanchate
181EC135

15 October 2020

Content

1. Packages Imported
2. Parse Program
3. Partitioning

Packages Imported

The following packages have been imported to implement and visualize Kernighan Lin algorithm for partitioning.

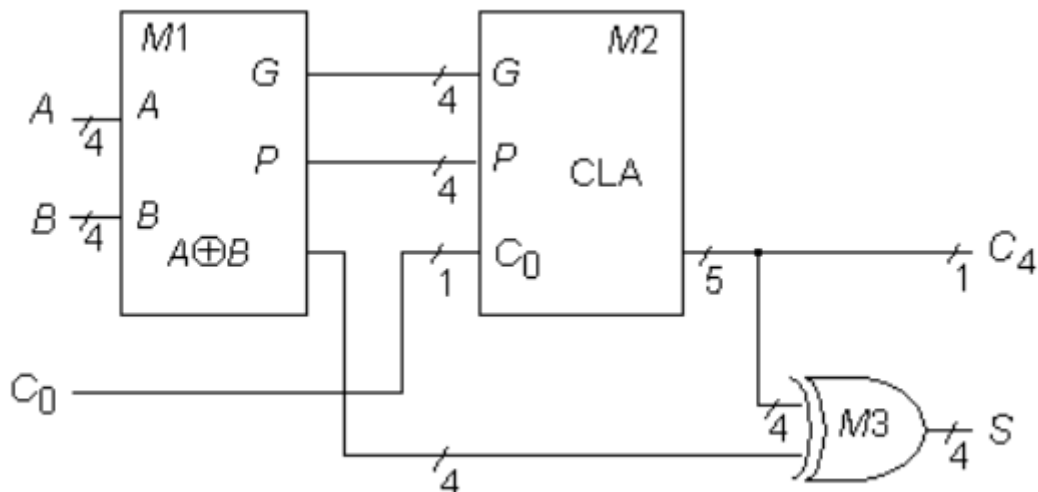
1. numpy
2. random
3. networkx
4. matplotlib

Parse Program

A parsing program (*parse.py*) has been created in order to parse and extract the circuit out of the netlist. This program is compatible with both ISCAS-85 and ISCAS-89 netlists.

Hence, I'll be able to do VLSI System Design Automation on **any of the netlists** given on the website. I'll be using [Fast Adder Circuit \(74283.isc\)](#) as an example throughout this project.

The circuit for Fast Adder Circuit is:

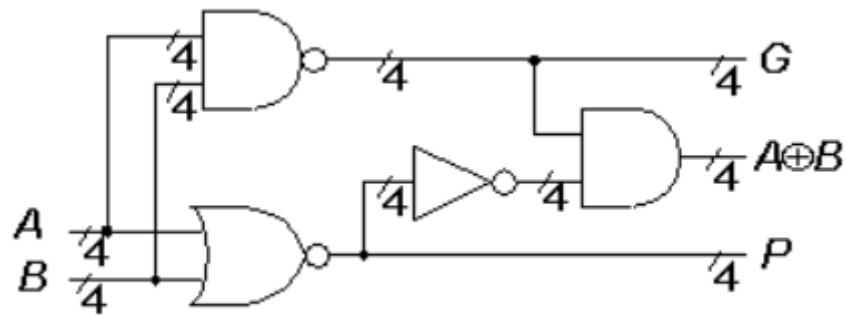


This circuit has:

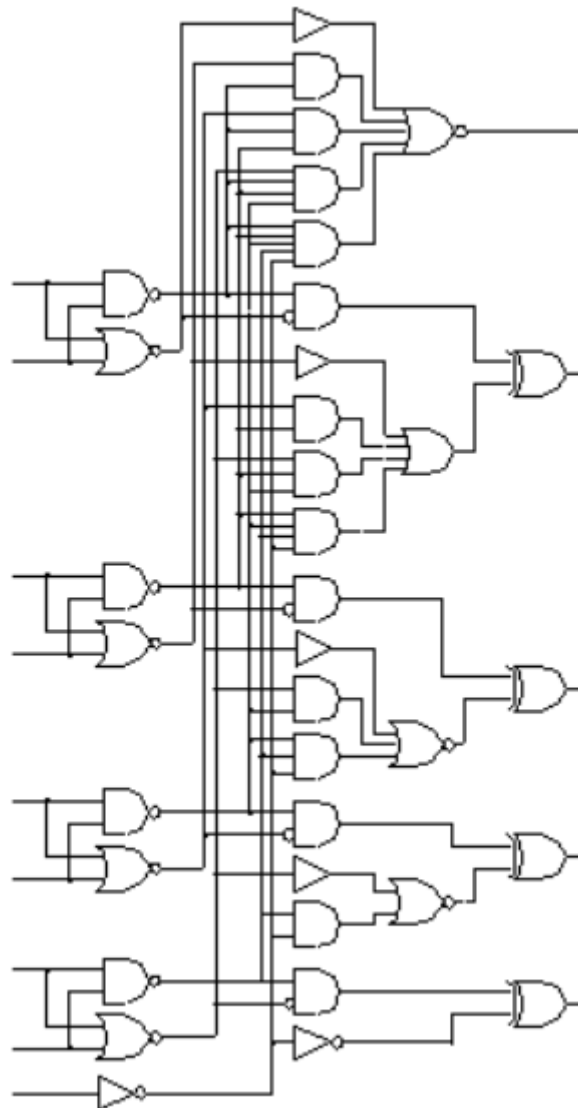
- 9 inputs
- 5 outputs
- 36 gates

The module M1 produces the generate, propagate, and XOR functions. The module M2 is similar to the 74182 (Carry Look-Ahead Circuit). The XOR word gate M3 produces the sum function.

The gate level schematic for module M1 is given below:



The gate level schematic of the entire circuit is given below:



Other netlists can be downloaded from [here](#). There are various files with *.isc* and *.bench* extensions which are ISCAS-85 and ISCAS-89 netlists respectively.

The `parse.py` program uses this file to generate required input for floor planning, partition, placement and routing programs.

Functions:

- **netadj85(filename)**: Parse function for ISCAS-85 netlist.
- **netadj89(filename)**: Parse function for ISCAS-89 netlist.

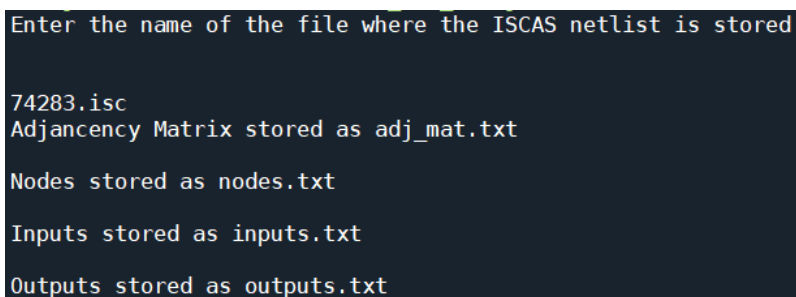
Input:

- **filename**: xyz.isc or xyz.bench

Output:

- **adj_mat.txt**: Contains the adjacency matrix denoting the connections between different nodes. The nodes are usually gates and sometimes a part of the logic (DFFs, etc)
- **nodes.txt**: Contains the list of all nodes of graph which will be created later. It contains nodes, type of nodes and number of inputs. This will help while creating the blocks in floorplanning.
- **inputs.txt**: Contains the list of all input nodes of the graph.
- **outputs.txt**: Contains the list of all output nodes of the graph.

The following image shows a demonstration of how the `parse.py` works.

A terminal window with a dark background and light green text. It shows the command prompt asking for the file name, followed by the user input '74283.isc'. The program then outputs four lines indicating the successful storage of the adjacency matrix, nodes, inputs, and outputs into their respective .txt files.

```
Enter the name of the file where the ISCAS netlist is stored
74283.isc
Adjacency Matrix stored as adj_mat.txt
Nodes stored as nodes.txt
Inputs stored as inputs.txt
Outputs stored as outputs.txt
```


outputs.txt:

```
outputs.txt x
1 97gat
2 101gat
3 102gat
4 103gat
5 104gat
6
```

nodes.txt:

```
nodes.txt x
1 ['26gat', 'not', 1]
2 ['32gat', 'nand', 2]
3 ['38gat', 'nor', 2]
4 ['41gat', 'nand', 2]
5 ['49gat', 'nor', 2]
6 ['53gat', 'nand', 2]
7 ['61gat', 'nor', 2]
8 ['66gat', 'nand', 2]
9 ['72gat', 'nor', 2]
10 ['78gat', 'not', 1]
11 ['79gat', 'not', 1]
12 ['80gat', 'not', 1]
13 ['81gat', 'not', 1]
14 ['82gat', 'and', 2]
15 ['83gat', 'and', 3]
16 ['84gat', 'and', 4]
17 ['85gat', 'and', 5]
18 ['86gat', 'and', 2]
19 ['87gat', 'and', 2]
20 ['88gat', 'and', 3]
21 ['89gat', 'and', 4]
22 ['90gat', 'and', 2]
23 ['91gat', 'and', 2]
24 ['92gat', 'and', 3]
25 ['93gat', 'and', 2]
26 ['94gat', 'and', 2]
27 ['95gat', 'and', 2]
28 ['96gat', 'not', 1]
29 ['97gat', 'nor', 5]
30 ['98gat', 'nor', 4]
31 ['99gat', 'nor', 3]
32 ['100gat', 'nor', 2]
33 ['101gat', 'xor', 2]
34 ['102gat', 'xor', 2]
35 ['103gat', 'xor', 2]
36 ['104gat', 'xor', 2]
```


Partitioning (Kernighan Lin Algorithm)

Kernighan Lin is probably the best and one of the easiest way to implement partitioning. It proposes a graph which is a bi-sectioning algorithm which starts with a random initial partition and then uses pairwise swapping of vertices between partitions, until no improvement is possible. One of the problems with the K-L algorithm is the requirement of prespecified sizes of partitions. This algorithm based on group migration are used extensively in partitioning VLSI circuits.

It starts by initially partitioning the graph $G = (V, E)$ into two subsets of equal sizes. Vertex pairs are exchanged across the bisection if the exchange improves the cutsize. The cost of partition is called the cutsize, which is the number of hyperedges crossing the cut. The above procedure is carried out iteratively until no further improvement can be achieved.

The algorithm is given below:

```
Algorithm KL
begin
  INITIALIZE();
  while( IMPROVE(table) = TRUE ) do
    (* if an improvement has been made during last iteration,
    the process is carried out again. *)
    while ( UNLOCK(A) = TRUE ) do
      (* if there exists any unlocked vertex in A,
      more tentative exchanges are carried out. *)
      for ( each  $a \in A$  ) do
        if ( $a = \text{unlocked}$ ) then
          for( each  $b \in B$  ) do
            if ( $b = \text{unlocked}$ ) then
              if ( $D_{\max} < D(a) + D(b)$ ) then
                 $D_{\max} = D(a) + D(b)$ ;
                 $a_{\max} = a$ ;
                 $b_{\max} = b$ ;
              TENT-EXCHGE( $a_{\max}, b_{\max}$ );
              LOCK( $a_{\max}, b_{\max}$ );
              LOG(table);
               $D_{\max} = -\infty$ ;
            ACTUAL-EXCHGE(table);
          end.
```

The algorithm is implemented in Python and produces partitions with least amount of cutsize possible. The time complexity of Kernighan-Lin algorithm is $O(n^3)$. The complexity of this algorithm is considered too high even for moderate size problems.

It starts with a randomly assigning the nodes into two groups obtained from the parsing program. It is then passed to *kernighanlin()* function which continuously iterates the *bestswap()* function. If the cutsize is converging, it breaks out of the *kernighanlin()* function. The *bestswap()* function iterates for all the number of pairs available in which nodes are exchanged and looked for the lowest cutsize by subtracting the previous cutsize with the current iteration. If the difference is seen to be the best amongst all iterations, the partitions for that iteration is taken into consideration. But during this practise, we might also take some amount of worse iterations during the search for the best iteration. All the pairs are swapped exactly once during *bestswap()* iterations. Thus, we can say that this is an example of a heuristic algorithm.

The characteristics of the Kernighan Lin implementation of partitioning in Python is given below:

Functions:

- **cutsize():**
 - **Inputs:** partition1, partition2, adjacency matrix
 - **Output:** cutsize
- **bestswap():**
 - **Inputs:** partition1, partition2, adjacency matrix, number of pairs
 - **Outputs:** Lowest cutsize among the iterations, the partitions corresponding to that cutsize.
- **kernighanlin():**
 - **Inputs:** partition1, partition2, adjacency matrix, number of pairs
 - **Outputs:** Final cutsize and partitions

Inputs:

- Adjacency Matrix file generated from the parsing program.
- Nodes from the parsing program. (For visualization only)

Outputs:

- Initial partitions with cutsize.
- Final partitions with cutsize.

The following image shows the output of the *kernighanlin.py* for the Fast Adder Circuit (74283.isc).

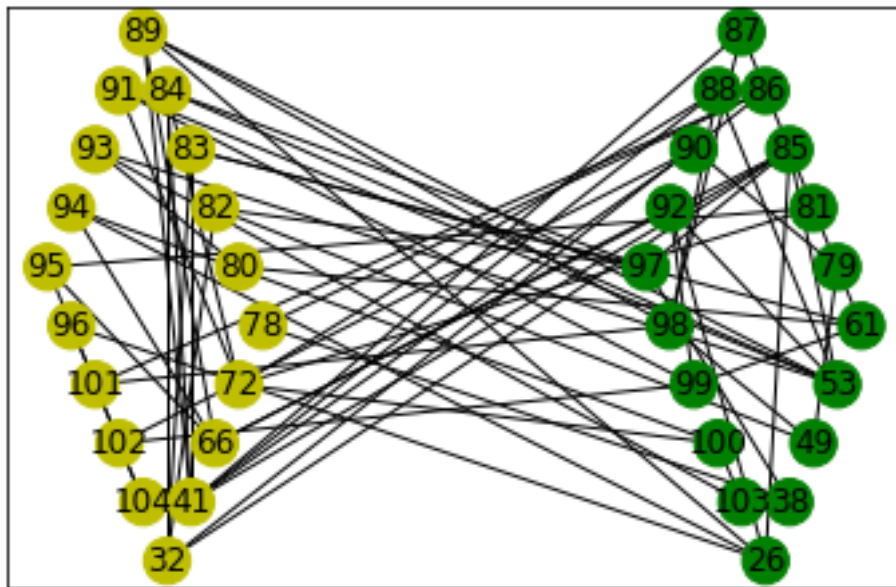
```
Enter the name of the file containing the required netlist:

74283.isc

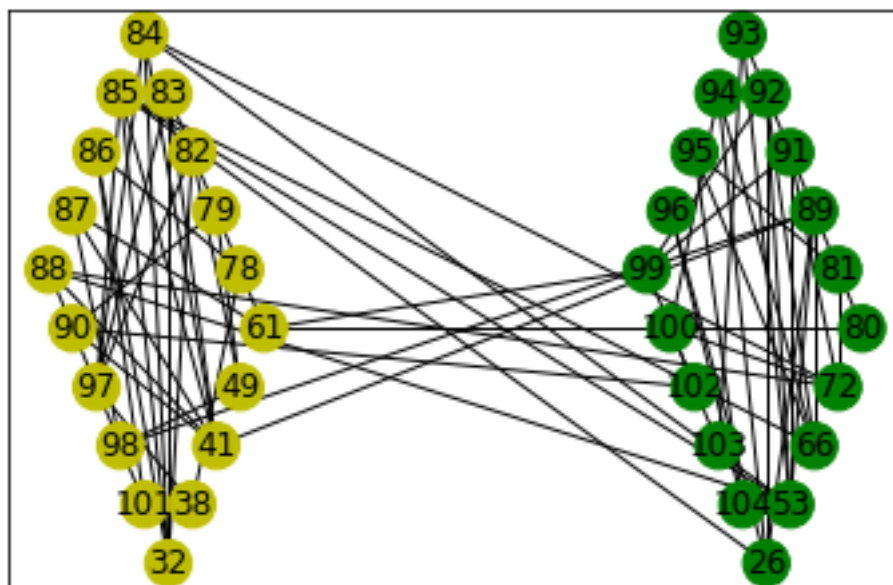
Initial Cut size is : 35
Initial Partition 1 : ['32gat', '41gat', '66gat', '72gat', '78gat', '80gat', '82gat',
'83gat', '84gat', '89gat', '91gat', '93gat', '94gat', '95gat', '96gat', '101gat', '102gat',
'104gat']
Initial Partition 2 : ['26gat', '38gat', '49gat', '53gat', '61gat', '79gat', '81gat',
'85gat', '86gat', '87gat', '88gat', '90gat', '92gat', '97gat', '98gat', '99gat', '100gat',
'103gat']
Final Cut size is : 12
Final Partition 1 : ['32gat', '38gat', '41gat', '49gat', '61gat', '78gat', '79gat', '82gat',
'83gat', '84gat', '85gat', '86gat', '87gat', '88gat', '90gat', '97gat', '98gat', '101gat']
Final Partition 2 : ['26gat', '53gat', '66gat', '72gat', '80gat', '81gat', '89gat', '91gat',
'92gat', '93gat', '94gat', '95gat', '96gat', '99gat', '100gat', '102gat', '103gat',
'104gat']
```

A graph visualization of the same has also been implemented. The partitions are represented with two different colors. The number of edges crossing from one group to another can easily be visualised and concluded that the cutsize has drastically been reduced.

Before Partitoning (Cutsite = 35):



After Partitioning (Cutsite = 12):



Thus, partitioning using KL algorithm has been successfully implemented.