EE6094 CAD for VLSI Design

Programming Assignment 4: Routing

Student ID: 108501023
Student Name: 李品賢

Compile, Execute and Verification

1. Pulls the source code, i.e., *108501023_PA4.cpp*, *Makefile*, *case0.txt*, *case7.txt*, *case8.txt* and *checker* into the workstation folder.

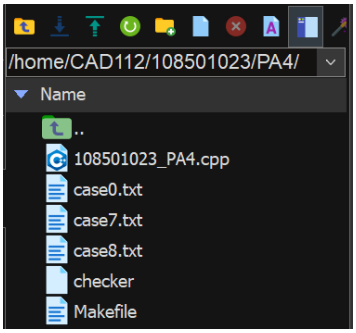

Fig. 1

2. Uses *Makefile* as a trigger point to run the *108501023_PA4.cpp* program, and then output files *out0.txt* / *out7.txt* / *out 8.txt* are generated.

```
[s108501023@eda359_forclass ~/PA4]$ make all
[s108501023@eda359_forclass ~/PA4]$ make run input=case0.txt output=out0.txt
[s108501023@eda359_forclass ~/PA4]$ make run input=case7.txt output=out7.txt
[s108501023@eda359_forclass ~/PA4]$ make run input=case8.txt output=out8.txt
[s108501023@eda359_forclass ~/PA4]$ make clean
```

Fig. 2

- make all
- make run intput=case0.txt output=out0.txt
- make run intput=case7.txt output=out7.txt
- make run intput=case8.txt output=out8.txt
- make clean

3. Uses checker to check whether output files fits the standard output format.

- ./checker out0.txt case0.txt
- ./checker out7.txt case7.txt
- ./checker out8.txt case8.txt

Completion

All three cases are successfully passed the checker. The following three figures (Fig. 3, Fig. 4, Fig. 5) are results.

	case0	case7	case8
Completion	O	O	O
# of track	3	3	4

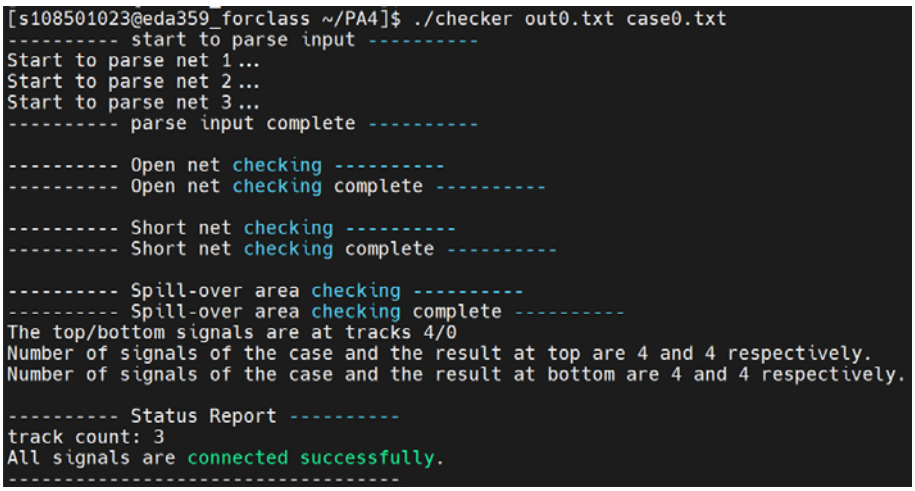

Fig. 3

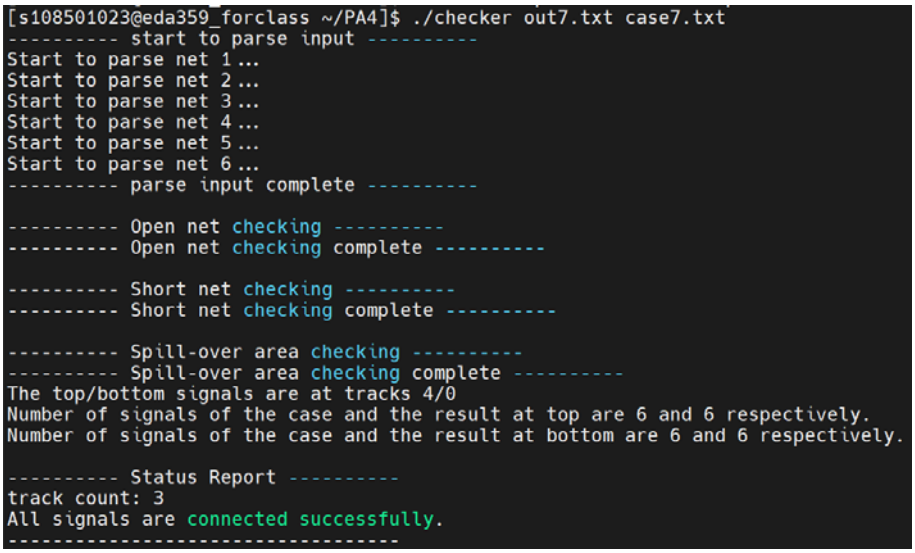

Fig. 4

```
[s108501023@eda359_forclass ~/PA4]$ ./checker out8.txt case8.txt
---------- start to parse input ----------
Start to parse net 1...
Start to parse net 2...
Start to parse net 3...
Start to parse net 4...
Start to parse net 5...
Start to parse net 6...
---------- parse input complete ----------

---------- Open net checking ----------
---------- Open net checking complete ----------

---------- Short net checking ----------
---------- Short net checking complete ----------

---------- Spill-over area checking ----------
---------- Spill-over area checking complete ----------
The top/bottom signals are at tracks 5/0
Number of signals of the case and the result at top are 9 and 9 respectively.
Number of signals of the case and the result at bottom are 8 and 8 respectively.

---------- Status Report ----------
track count: 4
All signals are connected successfully.
-----------------------------------
```

Fig. 5

Data structure

Due to the simplicity of basic left edge algorithm, the only slightly complicated data structure is vertical constraint graph (VCG). This graph focuses on the precedence of each node, so I use the simple 1-D adjacent graph data structure to represent VCG.

Fig. 6 shows a channel routing and also the VCG of it. This channel could be represented by adjacent graph in Fig.7. Each head node records the number of its predecessor and then points to its successors.

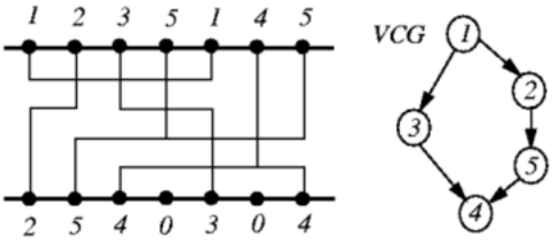

Fig. 6

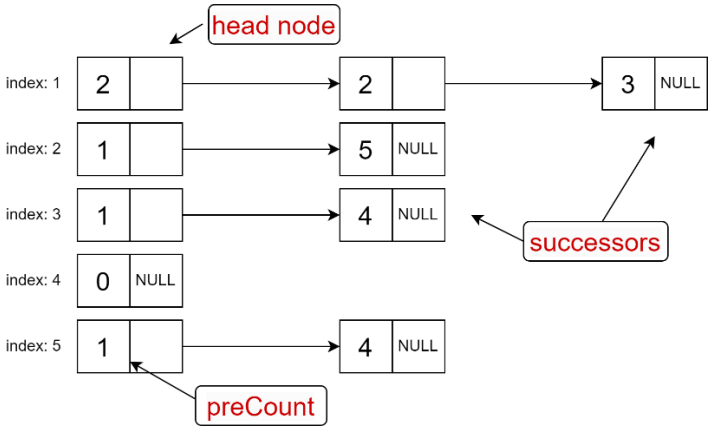

Fig. 7

Algorithm

In this program, the main algorithm is basic left edge algorithm (Fig. 8). It's pretty similar to solving the activity selection problem, and the only difference is that the basic left edge algorithm need to repeat until all horizontal intervals are scheduled.

In one iteration, one track is formed. In a track, horizontal intervals without overlapping could be scheduled on it, and this is achieved by *watermark* variable. Finally, all horizontal intervals are scheduled, and this function terminates. Then, the only thing needed to do is output all tracks in standard format.

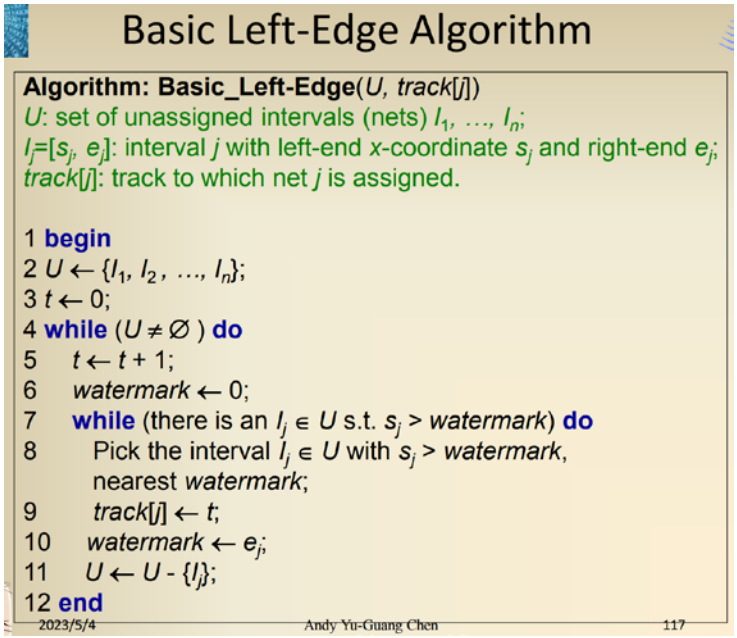

Fig. 8

Flow Chart

1. Overall program flow chart

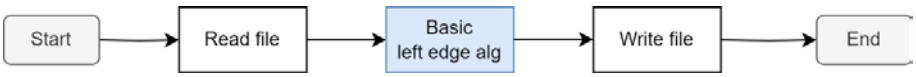

Fig. 9

2. Basic left edge algorithm flow chart

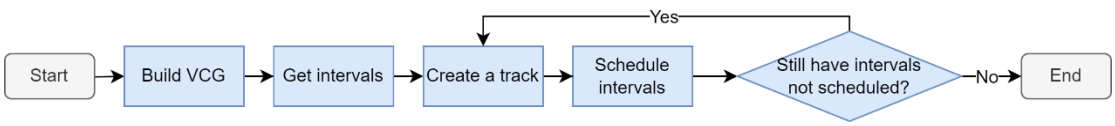

Fig. 10

Data structure in program

1. **class Pin** – indicates the information of each pins

Data type	Name	Purpose
bool	valid	Records if the pin exists in the input file
int	preCount	Records the predecessor count of pins
int	trackNum	Records the number of tracks of pins
Interval	I	Records the horizontal interval of pins
vector<int>	TOP_loc	Records the pin location on the top channel
vector<int>	BOT_loc	Records the pin location on the bottom channel
class Node *	ptr	Points to its successor

2. **class Node** – indicates the precedence of each pins

Data type	Name	Purpose
int	pinNun	Records the pin number of the successor
class Node *	ptr	Points to another successor of pins

3. **class Interval** – indicates the horizontal interval of each pin

Data type	Name	Purpose
int	pinNun	Records which pin number the horizontal interval belongs to
int	left	Records the starting location of the horizontal interval
int	right	Records the end location of the horizontal interval
bool	update	Records if the interval is scheduled

Important variables

Data type	Name	Purpose
vector<int>	TOP	Stores pins in the top channel
vector<int>	BOT	Stores pins in the bottom channel
vector< vector<Interval>>	track	Stores horizontal intervals in each track
vector<Interval>	I	Stores all horizontal intervals of pins

vector<Pin>	pin	Stores the information of each pin
int	columnCount	Stores the column count of the channel
int	ICount	Stores the horizontal interval count of all horizontal intervals

Important functions

(The list ignores the “Routung::” mark and arguments of functions)

1.	void build_graph()
	Uses this function to build the vertical constraint graph.
2.	void get_interval()
	Uses this function to collect each horizontal interval from the information of pins in top/bottom channel.
3.	void left_edge()
	Implements the basic left edge algorithm. First calls *build_graph()* function and *get_interval()* function, and then sorts horizontal intervals by its starting location. Afterward, creates a track and schedules the compatible horizontal intervals to it by using *watermark* variable. Repeats again and again until Icount is equal to 0.

Makefile

Because I use single .cpp file in this project, there is only one executable file created, i.e., *108501023_PA4.o*. The following is source code of *Makefile*.

```
1   # PA4 cpp compiler
2
3   .PHONY: all run clean
4   all: 108501023_PA4.o
5       @g++ -std=c++11 108501023_PA4.o -o exe
6   run:
7       @./exe $(input) $(output)
8   clean:
9       @rm *.o
10  108501023_PA4.o: 108501023_PA4.cpp
11      @g++ -std=c++11 -c 108501023_PA4.cpp
```

Fig. 11

Hardness

The hardest part is to build the vertical constraint graph, but actually it’s not quite difficult. I could imagine that the data structure of VCG would be very complex if I implement the program in dog leg version.

Suggestion

I am grateful for having this project, it helps me integrating data structure background into this project.

Reference

[1] 2023Spring_EE6094_CAD_Chapter11_Routing_20230504_0210

[2] https://www.youtube.com/watch?v=DRvNcOQbYV8&ab_channel=JohnReuben