FPGA Routing - Final Project of CENG 4120

# Update History

Mar 27:

1. Modified the format of node list: <node ID> **<node type>** <node length> … (see [Input format](#_kprgc3zaaipb)).
2. Added details of the ranking method (see [Benchmarks and Grading Scheme](#_2d0qte9krmbl)).
3. Device, benchmarks and the evaluator are released. (see [Benchmarks and Grading Scheme](#_2d0qte9krmbl)).

Mar 28:

1. Time limits are released (see [Benchmarks and Grading Scheme](#_2d0qte9krmbl)).

Apr 2:

1. Time limits for designs are decreased (see [Benchmarks and Grading Scheme](#_2d0qte9krmbl)).
2. Deadlines are released (see [Important deadlines](#_30f6xai8lkdz)).
3. The statement of optional alpha submission is released (see [Alpha submission (optional)](#_bw7ka7k3mdpe)).
4. Q&A updated.

Apr 3:

1. Change to use HPC14 as the test machine (see [Instruction to use CSE HPC Slurm](#_3r5wq0dtiw1m), please let TA know if you encounter problems).
2. Time limits are updated since the testing machine change to HPC14 (see [Benchmarks and Grading Scheme](#_2d0qte9krmbl)).

Apr 8:

1. Q&A updated.

Apr 9:

1. Q&A updated.

# Important deadlines

| Item | Date |
| --- | --- |
| Team formation | Apr 1, 23:59 |
| Alpha submission (optional) | Apr 25 23:59 |
| Final submission | May 3 23:59 |
| Presentation & Result released | May 6 |
| (More …) |  |

# Introduction

Routing is one of the most important and time-consuming phases in the compilation flow of Field-Programmable Gate Arrays (FPGAs). Modern FPGAs are composed of a vast array of logic resources (including look-up tables (LUTs), flip-flops (FFs), etc.) and routing resources (including wires and programmable interconnects) as seen in the figures below.

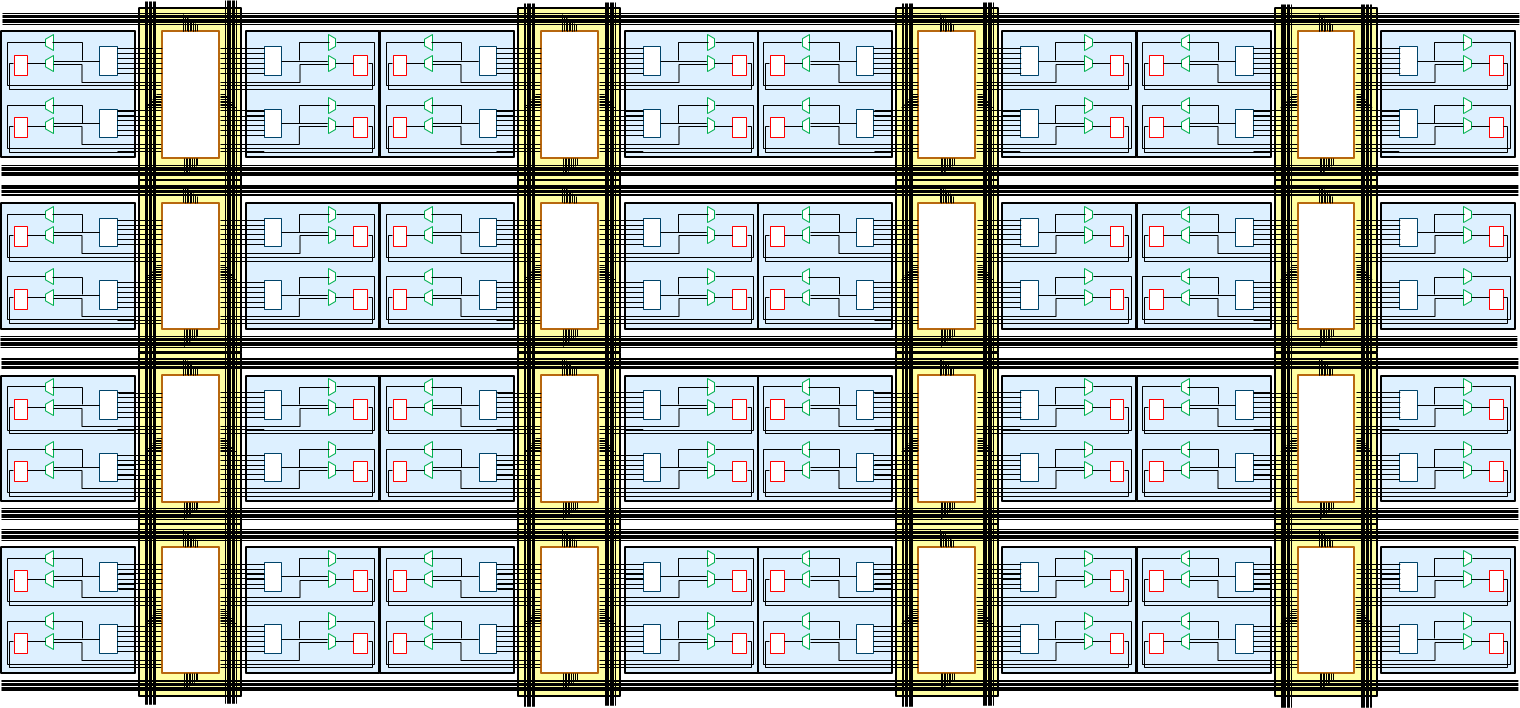


Figure: Hypothetical FPGA logic array of LUTs, flip flops and programmable interconnects

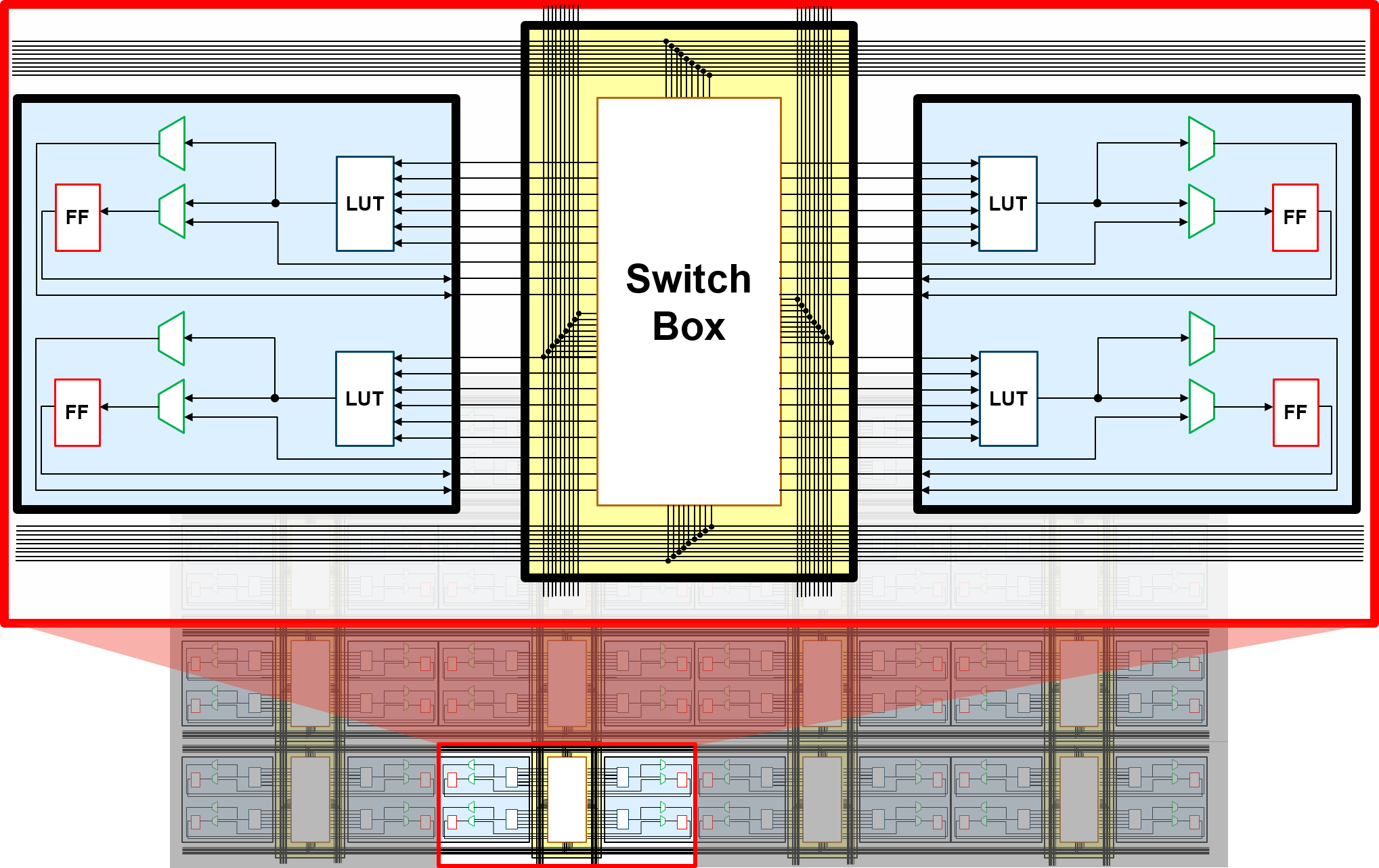


Figure: Close up view of replicated tiles of the logic array and interconnect

During the placement phase, the logic elements of the circuit design, such as logic gates and registers, are mapped onto specific logic resources on the FPGA board, such as LUTs and FFs. Then, the router utilizes the routing resources on the FPGA to connect these LUTs and FFs according to the placement solution and the netlist, ensuring that the routing resources are not shared among different nets. The outcome of the routing process has a direct and significant impact on the performance of the FPGA circuit. For example, an increase in wirelength can lead to high circuit delays and a corresponding reduction in operating frequency.

# Fundemental Concepts

## Tile

At an abstract level, Xilinx FPGA devices (including our target FPGA device xcvu3p) are constructed by assembling a grid of tiles. Xilinx FPGAs encompass various types of tiles, such as CLEs (Configurable Logic Elements), which contain logic resources like LUTs (Look-Up Tables) and FFs (Flip-Flops), and INTs (Interconnect Tiles), which house programmable switch boxes, etc. Each tile is an instance of a specific type and is assigned a unique name, with the prefix indicating the tile type and the suffix formatted as \_X#Y# (for example, INT\_X0Y0), denoting the tile's position. On the FPGA device, these tiles are arranged adjacently, as illustrated in the following diagram:

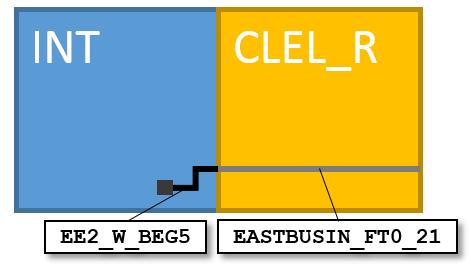


Figure: An example of two adjacent INT and CLE tiles ([Source](https://www.rapidwright.io/docs/_images/wire_abut.png))

## Routing resources

### Node

A node is a collection of electrically connected wires that spans one or more tiles. The figure below shows how four wires that abut among four tiles form a node:

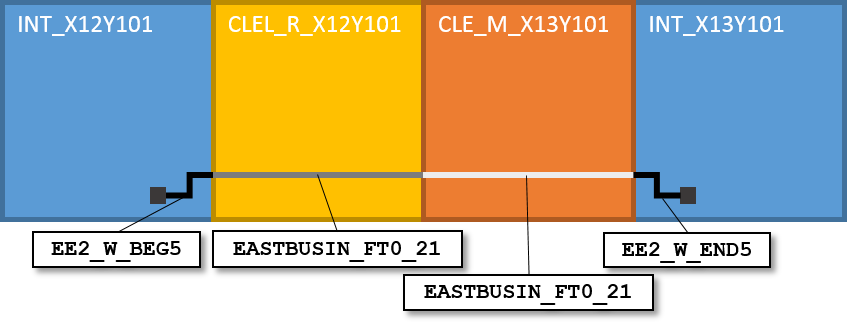


Figure: A node spanning 4 adjacent tiles ([Source](https://www.rapidwright.io/docs/_images/node.png))

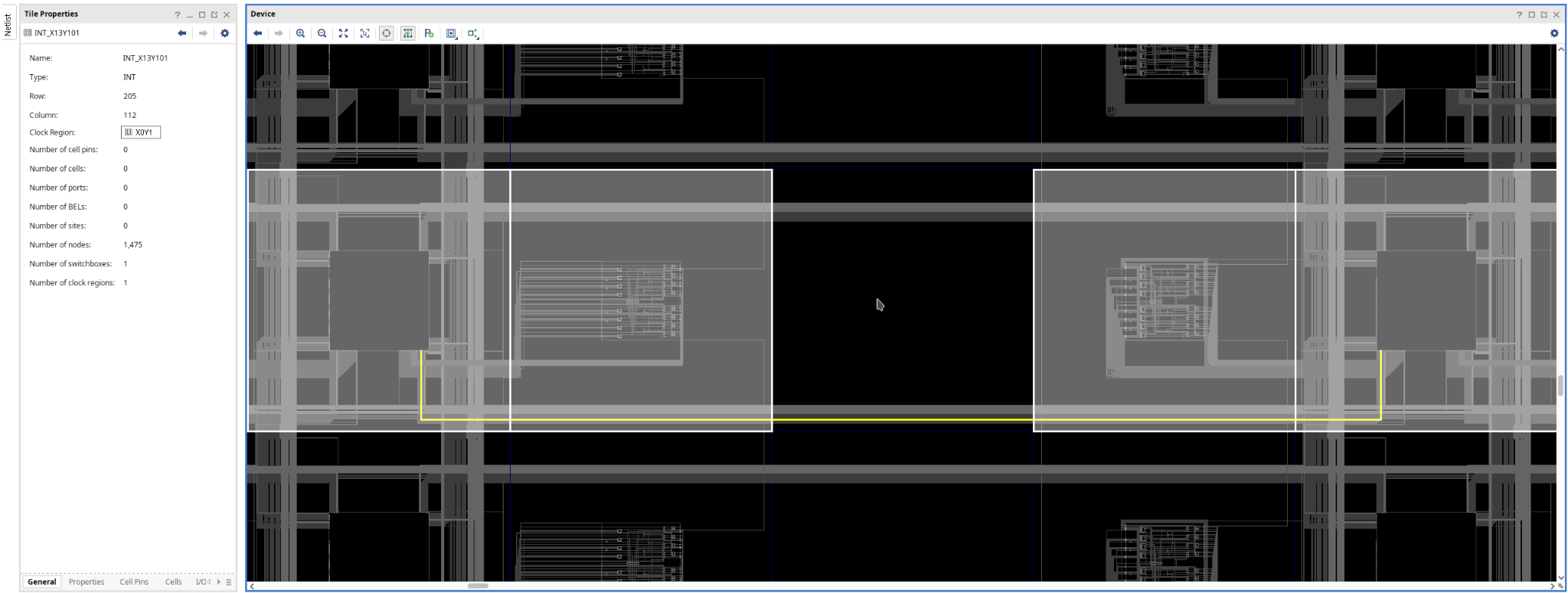


Figure: Vivado view (the yellow line is the above node INT\_X12Y101/EE2\_W\_BEG5)

Nodes are directional, meaning that electrical signals propagate from the beginning wire of a node (e.g., EE2\_W\_BEG5 in the figure) to its ​ending wire (e.g., EE2\_W\_END5). **Each node on the FPGA device will be abstracted as a node in the routing resource graph.**

### Programmable Interconnect Point (PIP)

Besides nodes, another kind of routing resources on the FPGA device are programmable multiplexers (muxes). A mux in a tile has multiple input wires, an output wire and a programmable selection bit. The input wires are usually end wires of upstream nodes, and the output wire is the beginning wire of a downstream node. By configuring the ​selection bit, users can determine which ​upstream node is connected to this ​downstream node. **In the routing resource graph, this *n*-input programmable mux will be abstracted as *n* directed edges (also called programmable interconnect points) as shown in the below figure.**

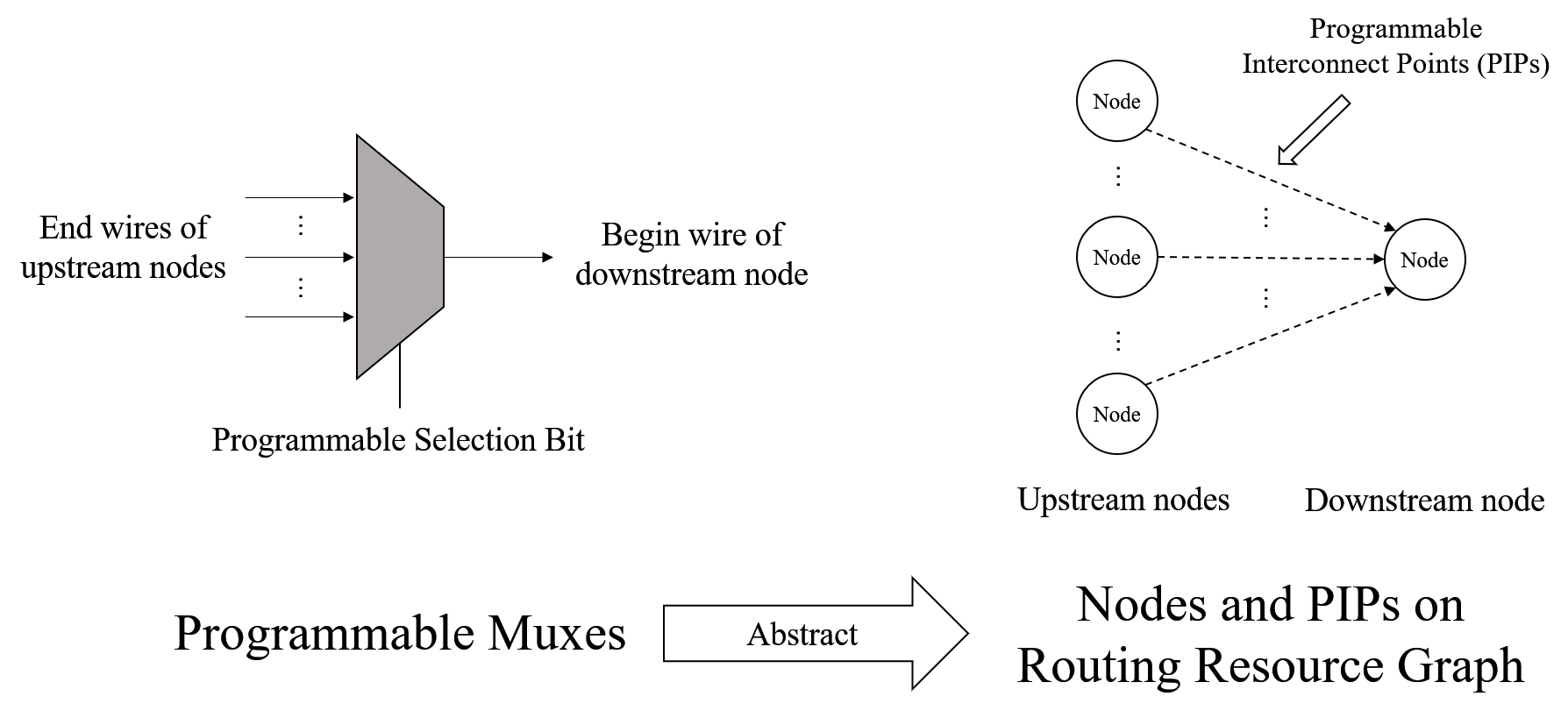


Figure: PIP abstract

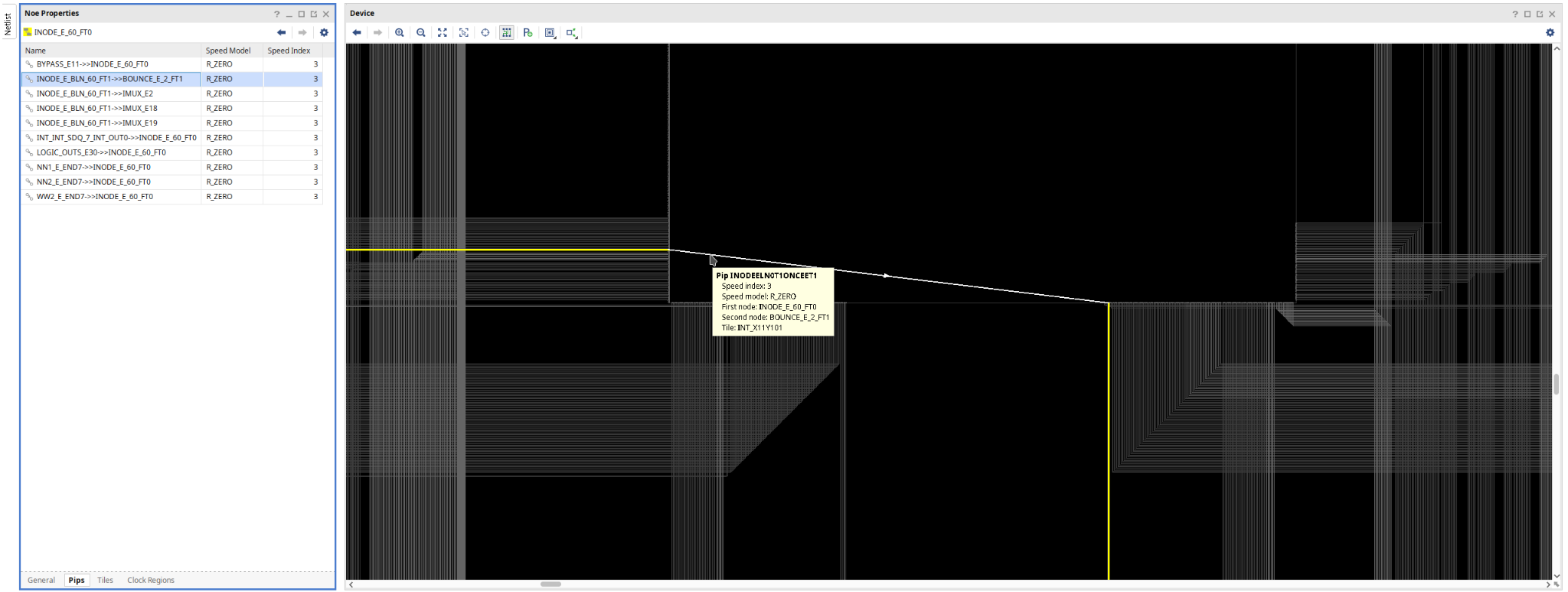


Figure: A PIP in Vivado view

## Net

Given the placement results, the logic elements are mapped to some logic resources on the FPGA device. We need to connect these logic resources according to the synthesised design. A **net** consists of a source node (which may be an output of some LUTs and FFs) and multiple sink nodes (which may be the inputs of some LUTs and FFs) driven by this source node. In the routing process, our router should generate a tree consisting of nodes and PIPs to interconnect these source and sink nodes.

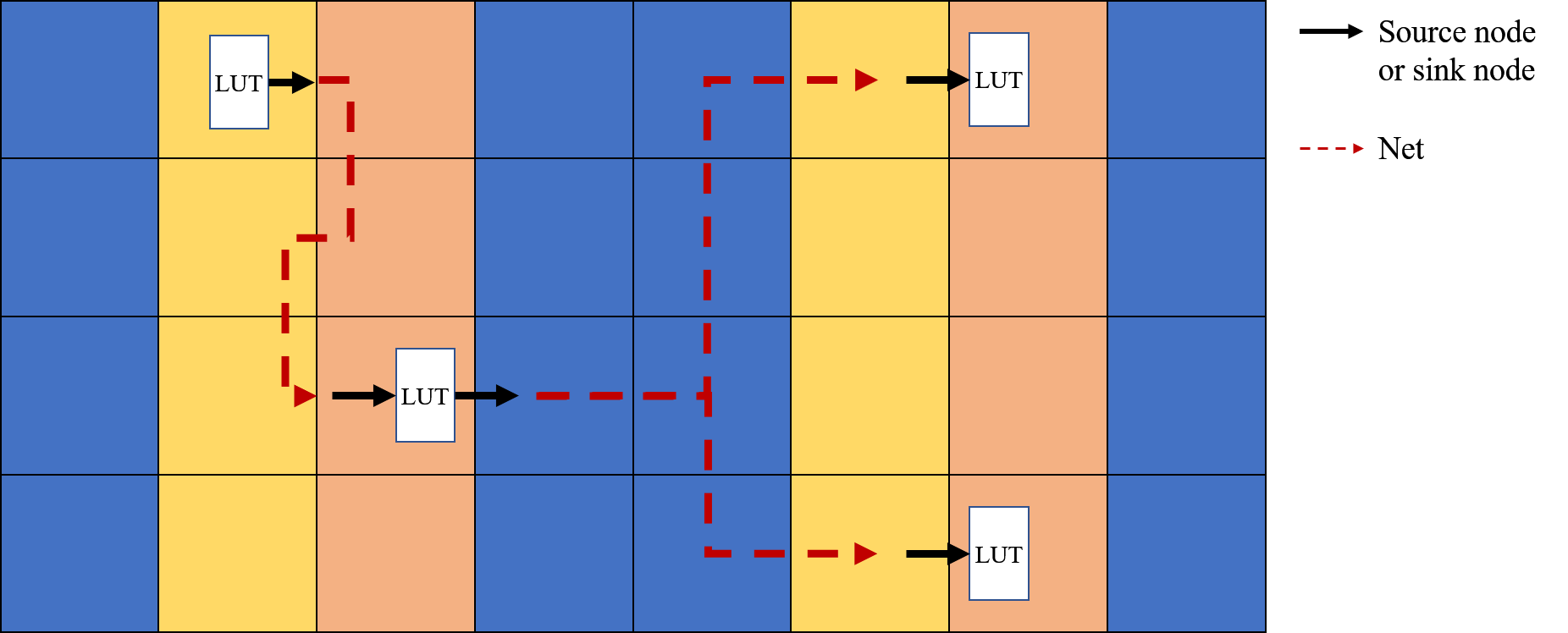


Figure: An example with two nets



Figure: A net (data\_out\_reg[5]\_rep\_\_0\_n\_0\_alias) in Vivado view

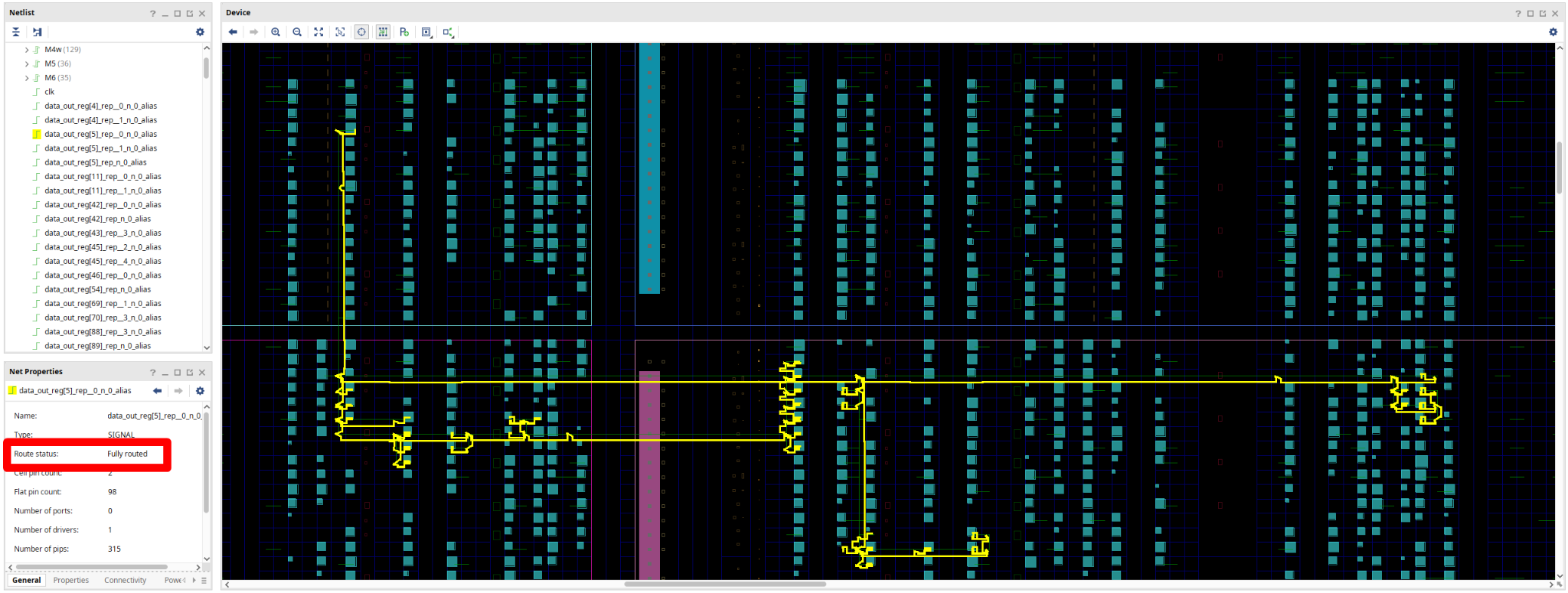


Figure: Routed net (data\_out\_reg[5]\_rep\_\_0\_n\_0\_alias)

Another important thing is, two different nets should not share nodes. Imagine that net A is driven by an output node of a LUT, whose output value is constant 1, and net B is driven by another output node of another LUT with output value 0. If a node is shared by these two nets, the value (and the electric potential) of this node is unknown. Nodes shared by different nets are called “**congested nodes**”, and a net using a congested node is called “**congested net**”

# Problem Description

Given the aforementioned concepts, we can define the FPGA routing problem as follows: FPGA routing problem P=<G, N>, where G is the routing resource graph and N is the netlist to be routed.

* The routing resource graph G=<V, E>, where V is the set of all nodes (each corresponds to a physical node on the FPGA device) and E is the set of edges (each correspond to a programmable interconnect point (PIP) on the FPGA device).
* The netlist N is a list of nets to be routed. Each net in the netlist N consists of a source node and multiple sink nodes.
* In the output of routing problem P:
  + For each net, the router should generate a tree that interconnects its source and sink nodes.
  + The trees of different nets should not share any nodes (all nets and nodes are not congested).

## Input format

The input of our FPGA router consists of two parts:

1. A simplified routing resource graph “xcvu3p.device”. This file describe G and consists of two parts:
   1. The first part is the list of all nodes on our target device (which describes V). The first line is the total number of nodes **n**. Each of following n lines describe one node and contains all information needed for routing, and the format is:

**<node ID> <node type> <node length> <begin x coordinate> <begin y coordinate> <end x coordinate> <end y coordinate> <node name>**

* 1. The second part is an adjacent list (which describes E). Each line starts with the ID of a node, followed by the IDs of its children nodes.

**<Parent Node ID> <Child Node ID 0> [<Child Node ID 1> [<Child Node ID 2> …]]**

1. Netlist <design name>.netlist
   1. The first line is the number of nets **m** in this design.
   2. Each of the following m lines describe a net and its format:

**<net ID> <net name> <source node ID> [<sink node ID 0> [<sink node ID 1> …]]**

## Output format

Your routing result of <design name>.netlist should be named as <design name>.route, which should organized as:

**<net ID> <net name>**

**<Parent node1 ID> <Child node1 ID>**

**<Parent node2 ID> <Child node2 ID>**

**…**

**<Parent node i ID> <Child node i ID>**

// an empty line

**<net ID> <net name>**

**<Parent node1 ID> <Child node1 ID>**

**<Parent node2 ID> <Child node2 ID>**

**…**

**<Parent node i ID> <Child node i ID>**

**…**

**<net ID> <net name>**

**<Parent node1 ID> <Child node1 ID>**

**<Parent node2 ID> <Child node2 ID>**

**…**

**<Parent node i ID> <Child node i ID>**

# Benchmarks and Grading Scheme

The benchmarks consist of 5 designs. Design 1 and 2 are two easy cases for debugging. Designs 3 and 4 are of medium difficulty. Design 5 consists of all signal nets in a real FPGA design.

For each design, we will rank the routing results of each team:

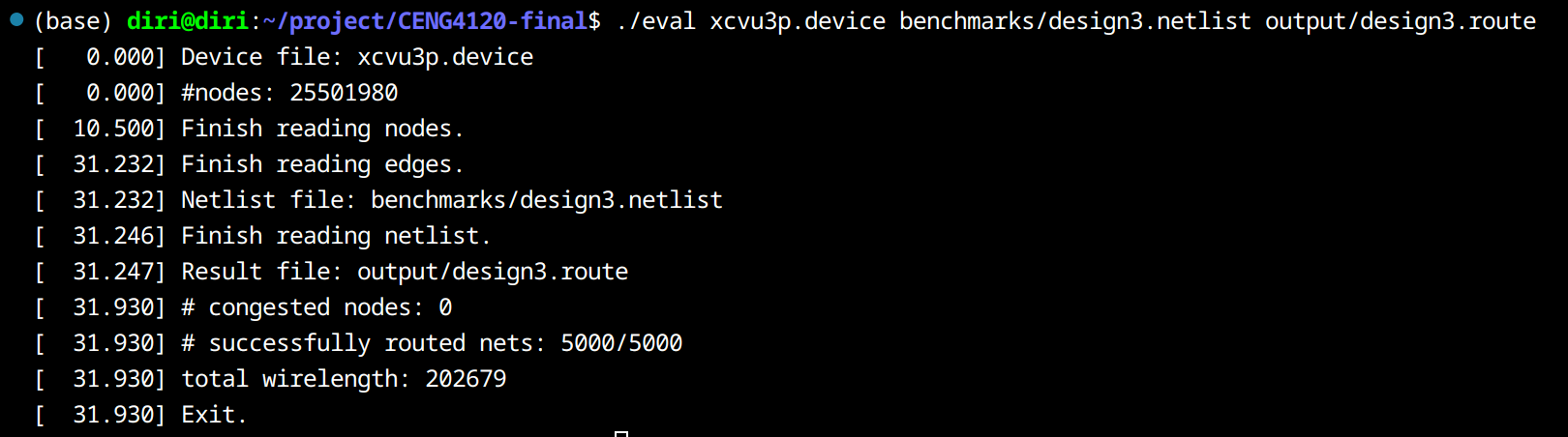
1. The team with congested nodes in the routing result or fails to finish routing within a limited time will be ranked last.
   1. The time limit for Designs 1-4 is **100** seconds, and for Design 5 it is **250** seconds. (They are approx. **2.5x** running time of the single-threaded sample router. For Design 5, the sample router uses **32s** to read input + **66s** to route + **1s** to save).
   2. Grading will be finished on server HPC11-14 (they have similar performance). You are encouraged to program and test your router on these servers to ensure your router will not break the time limits.
   3. You can find the instruction to use these servers in [Instruction to use CSE Slurm](#_3r5wq0dtiw1m).
2. Teams **without congested nodes** in the routing result will be ranked in **descending order of the number of successfully routed nets.** In case of a tie, the team with the smaller total wirelength of the routing result will be ranked first:

[](https://www.codecogs.com/eqnedit.php?latex=%5Ctext%7Btotal%20wirelength%7D%3D%5Csum_%7B%5Ctext%7BNet%20%7Dn%5Cin%20N%7D%5Csum_%7B%5Ctext%7BNode%20%7Dnode%5Cin%20n%7Dnode.length#0)

Finally, we calculate the weighted sum of the rankings of all designs and rank all teams in ascending order of this weight sum. The weights of 5 designs are 0.1, 0.1, 0.2, 0.2, and 0.4, respectively. Scores will be given according to the final ranking of your team (all team members will be given the same scores):

| Rank | Score | Rank | Score |
| --- | --- | --- | --- |
| Champion | 100% | 6th | 75% |
| Second-place | 95% | 7th | 70% |
| 3rd | 90% | 8th | 65% |
| 4th | 85% | 9th | 60% |
| 5th | 80% | 10th | 55% |
|  |  | Other teams | 50% |

## Download links

* benchmarks (zipped file): <https://github.com/ippan-kaishain/CENG4120-2025-Final/releases/download/Released/benchmarks.tar.bz2>
  + Unzip command: tar -xjvf benchmarks.tar.bz2
* evaluator: <https://github.com/ippan-kaishain/CENG4120-2025-Final/releases/download/Released/eval>
  + Usage: ./eval <device> <netlist> <result>
  + If you find no permission, try: chmod +x ./eval
  + 
* device (zipped file): <https://github.com/ippan-kaishain/CENG4120-2025-Final/releases/download/Released/xcvu3p.tar.bz2>
  + Unzip command: tar -xjvf xcvu3p.tar.bz2

## Alpha submission (optional)

You can submit your solution before the alpha submission deadline (Apr 25 23:59) to TA. TA will test your router and release an **anonymous** ranking to let you know the performance of your router.

This alpha submission is **optional**. You can skip this and only submit the final submission without any deduction. Only the ranking of final submission will count in the grading scheme.

# Instruction to use CSE HPC Slurm

According to CSE technical team, undergraduates should use HPC Slurm instead of HPC7 and HPC8. We will move to these servers to ensure that you can test the runtime of your router. **(The time limits will be updated according to the runtime of sample router later)**

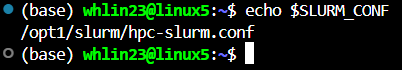
## Login

You use linux{1-10} as your login server. In this instruction, I use linux5.

## Setting ~/.bashrc

You can find a file called “.bashrc” in your home directory. Add a line “export SLURM\_CONF=/opt1/slurm/hpc-slurm.conf” to the end of this file. Then, execute “source ~/.bashrc” to ensure that this change is made in your current terminal.

You can verify it by executing “echo $SLURM\_CONF” and you should get the output



## Use slurm

Execute the command "srun -p hpc\_72h -w hpc**[11-14]** --cpus-per-task=<**#CPU threads**> --pty bash -i" and you will find (here I use HPC14 and apply 8 CPU threads):

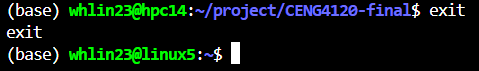


Now you create a job on Slurm and can run your router on HPC14 with up to 8 threads. Notice that:

1. You can create up to 4 jobs.
2. You can apply at most 80 CPU threads in total.
3. Your disk are shared by all linux[1-10] and HPC[1-16] machines so you can program on each machine that you can login to.

## Exit

Execute “exit”, then the job will be terminated and you will go back to your login machine.



For more information, see [Use Slurm to Submit Jobs – CSE Intranet](https://i.cse.cuhk.edu.hk/technical/gpgpu-hpc-service/slurm/) when you are connecting to CSE VPN.

# Q&A

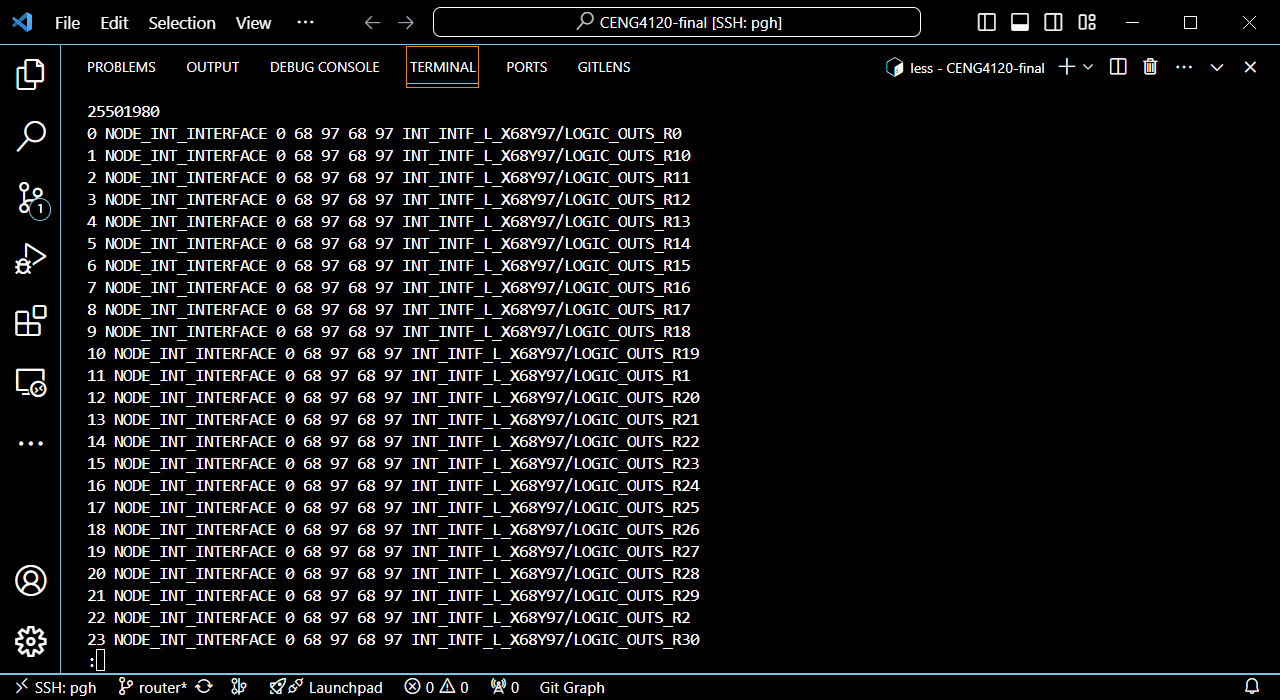
Q1: How to reduce the reading time?

A1: **Please consider this only if you have built a legal router and found your runtime is a little bit over the time limit.** I have two ideas:

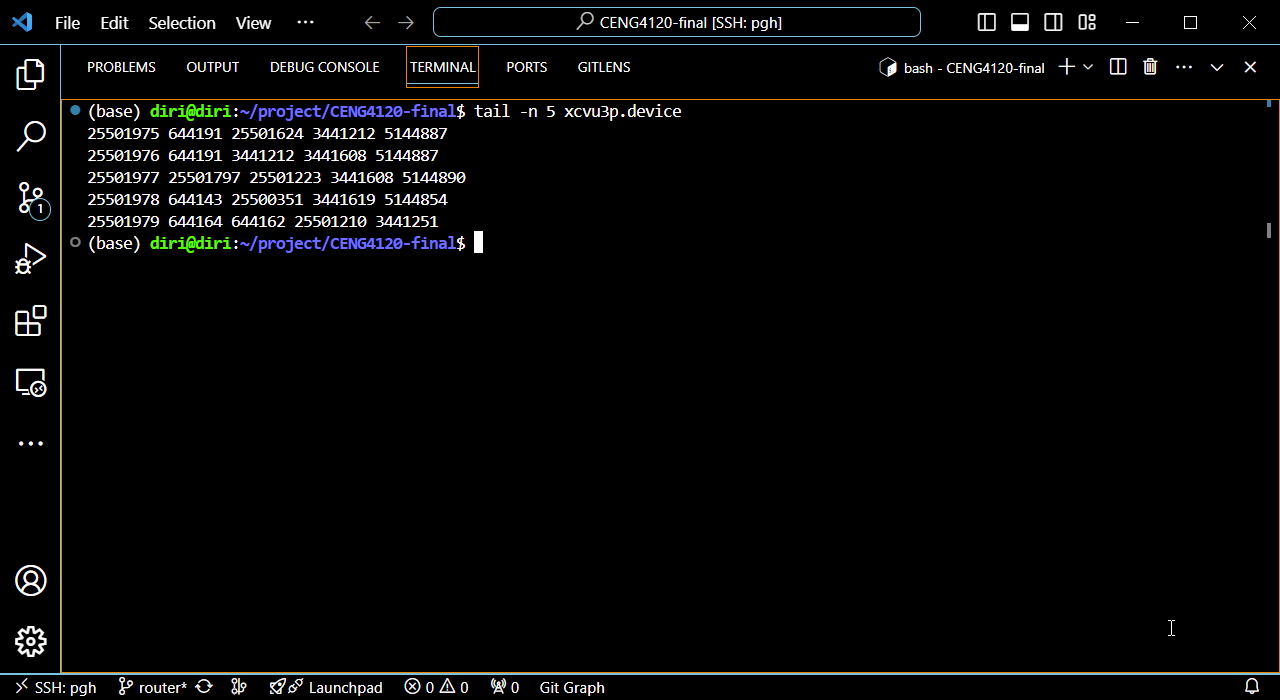
1. You can use multiple (up to 8) threads to parse your data.
2. You can try to “serialize” your data. Assume that you are using C++ and using a vector to store the parsed data of all your nodes. Your node data is a continuous piece of binary data in the memory. Try to dump it to a disk file, and next time you can load it from this disk file.

Q2: The device file is too large to open it in the text editor. How to open it faster?

A2: In a Linux computer, you can use the command “less <file>” to view it in the terminal. Then you can scroll up and down the window by your mouse. To quit, you can press “q”.



It may be hard to scroll down your window to see the edge list. You can use “tail -n <number of lines> <file>” to see the last <number of lines> lines of <file>:



Q3: When I tried to download xcvu3p.device, I found my disk quota was not enough?

A3: I put an xcvu3p.device in this path: /research/d1/gds/whlin23/public/xcvu3p.device and you can try to use it. This device file is readable but not writable.

Q&A4: **Some thoughts after the proposal presentation**

Glad to hear that you have many different ideas about this final project, and I have some suggestions for you. To finish this final project, we need to continuously repeat the process of: ​​’’Assess how far our router is from the target (runtime, wirelength, congestion, # routed nets … ) → Determine improvement directions → Implement changes”.

However, without an initial router, you’ll never know how far you are from your target. So, **my first suggestion is to ​​quickly build an initial router.** I’m glad to hear that some of you have already completed this step, but for those who haven’t, don’t worry—this is just a proposal presentation at the very beginning.

Many teams in this presentation mentioned ​​the time limits and expressed the need to improve their router's runtime. However, ​​very few teams actually **analyzed the breakdown of runtime in their own workflow**. With an estimated breakdown of the total runtime like:

[](https://www.codecogs.com/eqnedit.php?latex=%5Ctext%7BTotal%20runtime%7D%5Capprox%20%5Cleft(%5Csum_%7B%5Ctext%7Bsomething%7D%7D%5Csum_%7B%5Ctext%7Bsomething%7D%7D%5Ccdots%5Csum_%7B%5Ctext%7Bsomething%7D%7D%5Ctext%7BTime(something)%7D%5Cright)%2B%5Ctext%7Bnon-dominant%20part%7D#0)

, you will find your direction among these “something”s. This should also apply to other optimization objects, and this is **another suggestion: perform deeper analysis on your optimization objects**.