

ECE1756 Assignment 4

FPGA Routing Architecture

“All things are bound together. All things connect.”

— Chief Seattle

Assigned on Tuesday, November 28

Total marks = 12/100

Due on Wednesday, December 13 @ 11:59 pm

1 Objective

This assignment is intended to help you:

1	Develop experience in evaluating FPGA routing architectures using a full CAD flow.
---	--

2	Understand trade-offs in routing architecture design and optimization.
---	--

- You will run the VPR architecture evaluation tool on a set of different FPGA routing architectures, and determine which architectures are the most efficient.
- You will try to develop a routing architecture better than any of those given to you.

2 Deliverables

For this assignment, you are asked to hand in a **L^AT_EX-typed report** in PDF format using [this template](#) on Overleaf. You can also use this service to write your report if you do not want to install LaTeX on your machine. Your report should include the following:

1. Routing wire length study:

- (a) Graphs or tables showing the geometric average (over the 8 benchmarks) targeting the `k6_N10_40nm.xml` architecture file of: (a) minimum channel width, (b) low-stress routing area per tile, and (c) low-stress routing critical path delay vs. the length of the (single) routing wire type in the architecture. As described later, we are defining a *low stress* routing as one where the channel width is 30% higher than the minimum you found in part (a). To reduce “*CAD noise*” in your results, you should further average the results across multiple random seeds, or starting points, for the CAD tools; averaging over at least 5 seeds for each result is best. Your graphs or tables should include data points for architectures with all length 1 wires, all length 2, all length 4, all length 8, and all length 16 wires (5 architectures in total).
- (b) Explain your results. Why do the graphs have the shape they do?
- (c) Which routing wire length do you consider the best (gives the best combination of area and delay)?

2. Block to routing connectivity study:

- (a) For an architecture based off the `k6_N10_40nm.xml` file and with all length 4 wires, vary F_{cin} and F_{cout} for the logic blocks (CLBs). Include a table showing the geometric average (over the 8 benchmarks) of: (a) minimum channel width, (b) low-stress routing area per tile, and (c) low-stress routing critical path delay for $F_{cin} = F_{cout} = 0.15, 0.5, 1.0$ (3 architectures in total). Leave the F_{cin} and F_{cout} of the IO block at the default value of 0.15 for this experiment.

Note that this architecture has a full crossbar between the logic block input pins and the BLE/LUT input pins, as shown in Fig. 1. This means the VPR router is allowed to route a signal to any input of the logic cluster rather than to a specific input pin; the crossbar will get the signal to the right BLE. This crossbar inside the cluster also allows the VPR router to start a signal's routing from any logic block output, rather than the output corresponding to the specific LUT or FF that generates the signal. This is called logical equivalence, and it is specified in the architecture file by the lines shown below.

```
<input name="I" num_pins="40" equivalent="full"/> <output name="O" num_pins="10"
equivalent="instance"/>
```

- (b) Repeat part 2(a), but this time with the `k6_N10_sparse_crossbar_40nm.xml` architecture file, again using all length 4 wires. This architecture is the same as the earlier `k6_N10_40nm.xml` architecture except it uses a sparse crossbar for the local routing inside the logic block (clb), thereby reducing the switches needed to implement the local routing by 67%. When using an architecture with sparse local routing like this, it is beneficial to tell VPR to route signals not just to cluster inputs, but all the way through the local routing to the inputs of LUTs. The inputs of a LUT are logically equivalent (the router can re-arrange which input on a LUT each signal goes to as the CAD flow can change the LUT truth table to compensate), so routing all the way to the LUT inputs gives the router extra options that can help compensate for the less flexible local routing. This does come at the cost of a higher routing CPU time.
- (c) Explain your results – why do W_{min} , routing area and critical path delay vary (or not vary) with F_c in the way they do? Does the choice between a full crossbar and a partial crossbar impact the channel width and the best choice of F_c ?

3. Optimization study:

- (a) Try to find a routing architecture with a better area-delay product than the best architecture with a full local routing crossbar (i.e. based off `k6_N10_40nm.xml` you found in parts 1 and 2. You can change the various F_c values or use more than one type of wire (with different lengths for example). You could also change the switch pattern using the `switch_block` specification, and you can choose which points along a wire have routing switches and which do not. You can also change the electrical parameters of some or all of the wires. Assume that you can:
- make wires with R_{metal} reduced by 60%, but 20% more C_{metal} (by widening the wire at the expense of reducing the spacing between wires)
 - make wires with double the R_{metal} , but 40% less C_{metal} (by making the wire narrower and using the extra room to increase the spacing between wires)
 - you can also put up to **15%** of your wires on a higher metal layer with the same C_{metal} but only 25% of the R_{metal} of the wires in the given architecture file.

- (b) Describe your architecture, and explain why it works well.
- (c) Include the .xml file for this architecture in the appendix of your report.

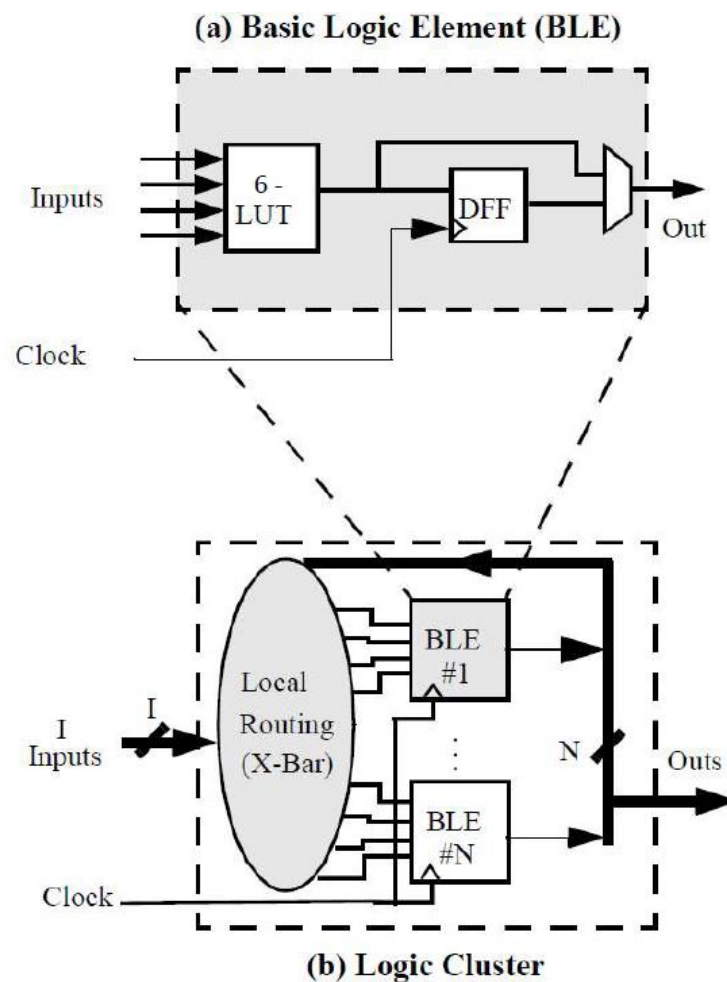


Figure 1: Logic block used in this assignment. It uses 6-input LUTs ($K=6$) contains ten LUTs per cluster ($N=10$), has 40 inputs ($I=40$) plus one clock. The `k6_N10_40nm.xml` logic block has a full-crossbar as its local routing, while the `k6_N10_sparse_crossbar_40nm.xml` logic block uses a sparse crossbar.

3 Detailed Specifications

First, get and build VTR:

```

1  git clone https://github.com/verilog-to-routing/vtr-verilog-to-routing.git
2  cd vtr-verilog-to-routing
3  git checkout ad7057f4bc1eaa3
4  git submodule init
5  git submodule update
6  make -j8 vpr

```

From Quercus, download *assign4.zip* to the machine that you are using (e.g. the UG machines) and unzip the file with the `unzip` command. The commands below assume you downloaded this file to the same directory in which you executed the `git clone` command to get VTR.

```
1 cd vpr
2 cp ~/Downloads/assign4.zip .      # Assuming you downloaded to ~/Downloads
3 unzip assign4.zip
```

The vpr executable you will need to run is at `./vpr` (i.e. in `vtr-verilog-to-routing/vpr/`). In the `./a4_benchmarks/` directory there are 8 circuits that contain only LUTs, FFs and IO blocks, and have been synthesized to target 6-input LUTs. These circuits are several thousand LUTs in size; they are smaller and simpler than what we would ideally use for architecture experiments, but that lets the CAD tools run quickly.

There are two architecture files in your current directory. `k6_N10_40nm.xml` describes an architecture where each logic block consists of ten 6-LUTs and 10 FFs (i.e. $N = 10$), with 40 inputs (i.e. $I = 40$) to the logic block, and a full crossbar between the cluster inputs and the look-up table inputs, as shown in Fig. 1. The only difference in the `k6_N10_sparse_crossbar_40nm.xml` architecture is that its logic blocks use a sparse crossbar, reducing local routing switches by 80%.

To complete the lab you will have to copy/modify `k6_N10_40nm.xml` and `k6_N10_sparse_crossbar_40nm.xml` to change the wire lengths and F_c values, as specified earlier.

All the results you report should be geometric averages over the 8 benchmark circuits. The geometric average is chosen to equally weight all 8 benchmarks; see [1] for details on why the geometric average is best for such a comparison. Additionally, if you have a fast enough computer or enough CPUs that the compute time does not become onerous, you will obtain smoother results with less CAD noise if you average your results over multiple CAD starting points, or random seeds. Use the `--seed <int>` option in VPR to specify a different starting point for the random number generator, which will make the CAD tools obtain a different placement. The default is 1 and any integer is a valid and equally good option. Using 5 seeds is a good practice, again so long as you can tolerate the CPU time.

To compute routing areas and circuit critical paths, you will follow a two-step procedure:

1. Run VPR on a circuit to get the minimum number of tracks per channel required to route it.
2. Usually FPGAs have some spare routing, for most circuits. Consequently it is more realistic to measure routing area and critical path delay with some spare routing in the device (which we call a low stress routing). Run VPR with $1.3\times$ the minimum channel width found in part 1, using the option `--route_chan_width <int>`. For single-driver / unidirectional routing (which we are using in this lab), the routing channel width must be divisible by 2, so round off to the nearest even number.

The 3 numbers you must report from the VPR output are shown below, for a sample circuit `frisc`. Note that I have also specified a seed for the starting point of the placement (by changing this seed I can get different placements over which I can average results). In this case the **minimum channel width (W_{min}) is 82**. After obtaining this number, I run VPR again with a channel width 30% larger than this minimum ($1.3 \times 82 = 106$). From this second run, I obtain the inter-cluster routing area (per tile) of **6528 minimum-width transistor areas**, and the circuit delay (inverse of maximum frequency) which is **9.29382 ns**. Note that the inter-cluster routing area does not include the area of the local routing inside the cluster; to fully compare the full crossbar vs. sparse crossbar architectures you would have to add the area of the logic block and its local routing as well (these values are in the architecture file if you are curious).

```

1 // First find the minimum channel width
2 $ vpr k6_N10_40nm.xml a4_benchmarks/frisc --seed 2
3 ...
4 Best routing used a channel width factor of 82.
5 ...
6
7 // Now run again with a channel width 30% bigger than that
8 $ vpr k6_N10_40nm.xml a4_benchmarks/frisc --seed 2 --route_chan_width 106
9 ...
10 Circuit successfully routed with a channel width factor of 106.
11 ...
12 Routing area (in minimum width transistor areas)...
13   Total routing area: 2.61158e+06, per logic tile: 6528.
14 ...
15 Final critical path delay (least slack): 9.29283 ns, Fmax: 107.61 MHz
16 ...

```

When running the architecture with a sparse crossbar (`k6_N10_sparse_crossbar_40nm.xml`), additional parameters should be added to VPR to enable flat (combined inter- and intra-cluster) routing and increase the maximum signal rip-up and re-try iterations in order to fully exploit the connectivity inside the cluster. The run-time of these experiments can be higher than the experiments with the architecture that has a full crossbar inside the cluster.

```

1 // First find the minimum channel width
2 $ vpr k6_N10_sparse_crossbar_40nm.xml a4_benchmarks/frisc --seed 2 --flat_routing true --
   max_router_iterations 200
3 ...
4 Best routing used a channel width factor of 86.
5 ...
6
7 // Now run again with a channel width 30% bigger than that
8 $ vpr k6_N10_sparse_crossbar_40nm.xml a4_benchmarks/frisc --seed 2 --flat_routing true --
   max_router_iterations 200 --route_chan_width 112
9 ...
10 Circuit successfully routed with a channel width factor of 112.
11 ...
12 Routing area (in minimum width transistor areas)...
13   Total routing area: 3.58657e+06, per logic tile: 8966.44
14 ...
15 Final critical path delay (least slack): 9.10157 ns, Fmax: 109.871 MHz
16 ...

```

You should write a script or program to launch VPR on the 8 circuits (and ideally for several seed values), parse and average the results, as otherwise it is too much typing to complete this lab. Some pre-built scripts are available as part of the VTR project in `vtr-verilog-to-routing/vtr_flow/scripts` but these scripts are complex and it is probably simpler to write a new script than to use or adapt these scripts.

4 Gotchas and Troubleshooting

- VPR produces several output files for each circuit (named after that circuit), and by default these go in the working directory from which you started `vpr`. Do not run multiple copies of VPR on the same benchmark in the same working directory at the same time, or you can get errors as the various copies of VPR try to write to the same file simultaneously.

- Since both the routing graph (switch pattern) generator and routing algorithm in VPR are heuristics, they can get inconsistent answers for some circuits on some routing architectures. For example, occasionally a circuit will route at some minimum channel width (W_{min}) but fail to route when the channel is widened by 30% due the placement interacting with a poor switch pattern in some localized part of the chip. If this happens, just run a different seed of that circuit or remove that seed from the average results you compute for that circuit on that architecture.

5 Documentation and Further Reading

Documentation on the latest version VTR is at <https://docs.verilogtorouting.org/en/latest/>, and a useful quick start guide is in the latest documentation at <https://docs.verilogtorouting.org/en/latest/quickstart/>.

The documentation on VTR is large, and you definitely should not try to read it all! For this assignment, you will be changing routing architectures, so the following sections are helpful references:

- Specifying the routing wires in an architecture.xml file: you can select their length, R, C, and where along them they can connect to other wires. <https://docs.verilogtorouting.org/en/latest/arch/reference/#wire-segments>
- Changing the Fc values that control how many wires each block input or output can connect to: search for `<fc>` in <https://docs.verilogtorouting.org/en/latest/arch/reference/#physical-tiles>
- Changing the switch block pattern in an architecture.xml file: search for `<switch_block>` in <https://docs.verilogtorouting.org/en/latest/arch/reference/#top-level-tags>

You can visualize the FPGA architecture you are using and how a circuit is mapped to it by specifying the `--disp` option to `vpr`. See <https://docs.verilogtorouting.org/en/latest/vpr/graphics/> for information on how to use the graphics.

Reference [2] gives a good overview of VPR and many trade-offs in routing architecture; references [3] and [4] are also good overviews of VPR (but less detailed than that in [2]). Reference [5] describes the many changes made to VPR in the 8.0 release, references [6] and [7] describe the current VPR placer and router, respectively, and reference [8] describes the how combined inter and intra-cluster (flat) routing works in VPR.

6 How You Will Be Graded

Your grade out of 12 will depend on:

1. **The correctness and clarity of your routing wire length and block to routing connectivity studies: ~67%.** Your report should include the requested graphs or tables, along with clear and correct answers to the questions asked in these sections. Your graphs or tables should be show the data in a clear way and be discussed in the report text.
2. **The thoroughness of your architecture optimization study and the quality of the resulting architecture: ~33%.** This grade will depend on the breadth of architectures you

explore (a wider variety is better), the area-delay product of your best architecture, and your explanation of why this architecture works well.

References

- [1] P. Fleming & J. Wallace, “How Not to Lie with Statistics: The Correct Method to Summarize Benchmark Results,” *Communications of the ACM*, March 1986.
- [2] V. Betz et al, "Architecture and CAD for Deep-Submicron FPGAs", Kluwer, 1999.
- [3] V. Betz & J. Rose, “VPR: A New Packing, Placement and Routing Tool for FPGAs” FPL 1997.
- [4] A. Marquardt et al, “Timing-Driven Placement for FPGAs” FPGA 2000.
- [5] K. Murray et al, "VTR 8.0: High-Performance CAD and Customizable FPGA Architecture Modelling," *ACM TRET*S, Vol. 13, No. 2, June 2020, pp. 9:1 - 9:55.
- [6] M. Elgammal et al, “Learn to Place: FPGA Placement Using Reinforcement Learning and Directed Moves,” *IEEE Conf. on Field-Programmable Technology*,” 2020, pp. 85 - 93.
- [7] K. Murray et al, “AIR: A Fast but Lazy Timing-Driven FPGA Router,” *Asia-Pacific Design Automation Conference*, Jan. 2020, pp. 338 - 344.
- [8] A. Mohaghegh et al, “Tear Down the Wall: Unified and Efficient Intra- and Inter-Cluster Routing for FPGAs,” *Int. Conf. on Field Programmable Logic*, Sept. 2023, pp. 1 - 9.