RapidSmith 2 Introductory Exercises

This document contains several introductory learning exercises to help new users learn how to develop CAD tools in RapidSmith. Its goal is to help designers become familiar with the following:

1. How to use RapidSmith Device data structures (Tiles, Sites, BELs, etc.)
2. How to use RapidSmith Design data structures (CellDesigns, Cells, CellNets, etc.)
3. How to modify a RapidSmith netlist to add or remove logic
4. How to Place and Route a logical netlist
5. How to import Tincr Checkpoints into RapidSmith.
6. How to export a RapidSmith netlist to a Tincr Checkpoint

As you complete these exercises, read through the Tech Report documentation and look through the example programs in *edu.byu.ece.rapidSmith.examples*.

1 CLB Printer

The purpose of this exercise is to gain a better understanding of the device data structures in RapidSmith. Before starting, read the **Device** section of the Tech Report in the RapidSmith repository. It gives a good overview of Xilinx FPGA architecture and terminology, and introduces the RapidSmith data structures that you will need for this task. As you work through this exercise, it may be helpful to open the device browser in Vivado. Create a Java program to do the following:

- Load the Artix7 part "xc7a100tcsg324-3" into RapidSmith
- Find a tile of type "CLBLL_L" in the device (this can be done by tile name, or getting all tiles in the device of type CLBLL_L and choosing the first one).

```
// Load the Artix7 device
Device device =
    RSEnvironment.defaultEnv().getDevice("xc7a100tcsg324-3");

// Get a handle to a Tile object
Tile t = device.getTile("CLBLL_L_X2Y69");
```

- Print the following information (in any format you want)
  - The name of all Wires in the Tile
  - The name and type of all Sites in the Tile
  - For each Site, print the Site Pins, BELs, and Wires within the Site

- Create a function that takes as input a Wire object *sourceWire* and an integer *numHops*, and returns a list of Wires that are *numHops* wires away from source wire. For example, if 1 is passed into the numHops parameter then all the wires directly connected to the source wire (through a PIP) should be returned. Only traversing a PIP wire connection counts as taking a hop.

```
public Set<Wire> getWiresHops(Wire sourceWire, int numHops) {
 Set<Wire> sinkWires = new HashSet<Wire>();

 // Your code here

 return sinkWires;
}
```

2 Design Printer

The purpose of this exercise is to gain a better understanding of the design data structures in RapidSmith. You will also learn how to load a Tincr Checkpoint (which represents a Vivado design) into RapidSmith. Before starting, read the **Designs** section of the Tech Report in the RapidSmith repository. It gives a good overview of the netlist data structures and how to use them. Create a Java program to do the following:

- Load the *cordic.tcp* Tincr checkpoint into RapidSmith (this design has been placed in Vivado, but not routed). This checkpoint is located in the *exampleVivadoDesigns* directory of the repository.
- Print the following about the loaded design (in any format you want):
  - The name of the design
  - The number of cells and nets in the design
  - For each cell, print its (1) name, (2) library cell type, (3) cell pins (with direction), (4) properties, and (5) BEL placement location (if any)
  - For each net, print its (1) name, (2) net type, (3) source cell pin(s), and (4) all sink cell pins
- Create a function to print a `CellNets` INTERSITE route tree to the console in a meaningful way (it is a tree data structure so you will need to represent branching). Make sure to distinguish between PIP wire connections and non-PIP wire connections. Using your function, print the `RouteTree` for the net "q_OBUF[0]" in the *count16.tcp* design. Verify that it matches the route tree in Figure 1.

```
// Load the count16.tcp checkpoint into RapidSmith
String tcpPath = RSEnvironment.defaultEnv()
                 .getEnvironmentPath()
                 .resolve("exampleVivadoDesigns")
                 .resolve("count16.tcp").toString();

TincrCheckpoint tcp = VivadoInterface.loadTCP(tcpPath);
CellDesign design = tcp.getDesign();
CellNet net = design.getNet("q_OBUF[0]");

// call your RouteTree function here
printRouteTree(net);
```

```
// optionally, you can print a graphical representation of the RouteTree for
// comparison
DotFilePrinter.getRouteTreeDotString(net);
```

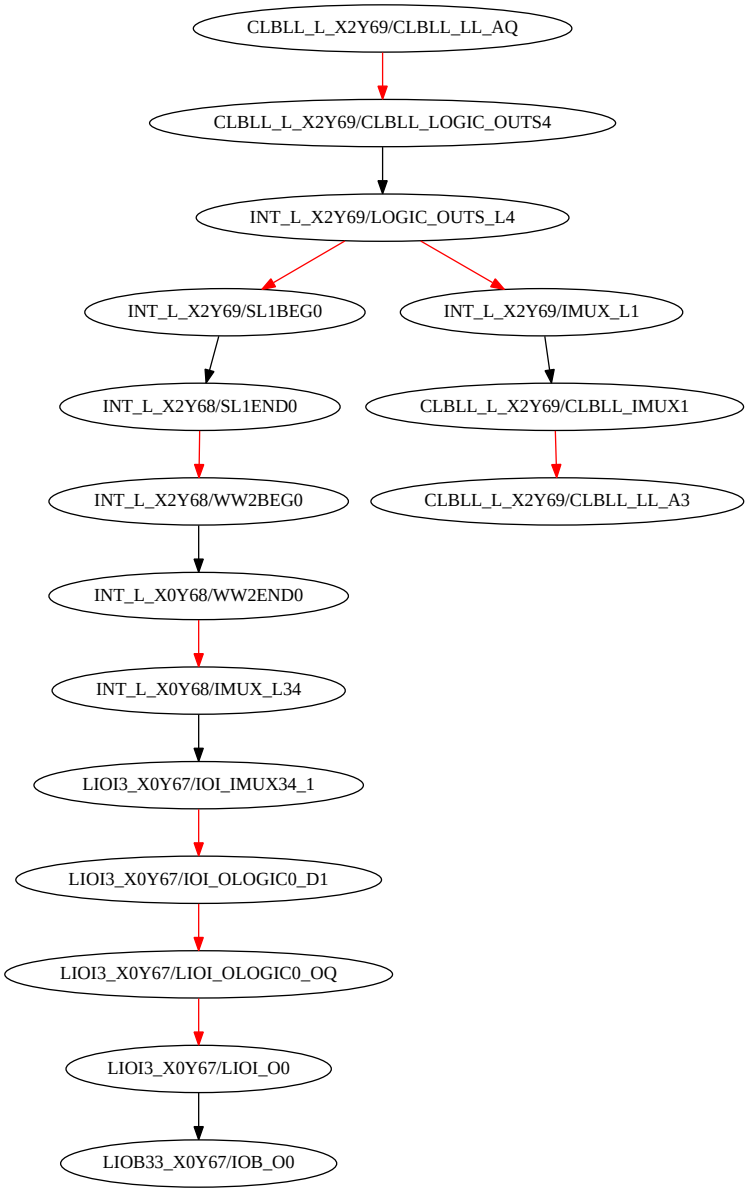

Figure 1: RouteTree visualization for the CellNet "q_OBUF[0]" in the *count16.tcp* design. This can be used to verify that your RouteTree printer function is working correctly.

3 Full Adder

The purpose of this exercise is to learn how to modify an existing netlist, or create a new netlist from scratch. Using LUTs and FF cells (FDRE) in RapidSmith, create a full adder circuit with registered outputs (Figure 2). Include a clock signal, but tie the reset signal on the FF cells to ground. The Xilinx series7 CLB documentation https://www.xilinx.com/support/documentation/user_guides/

ug474_7Series_CLB.pdf may be helpful to learn more about LUTs and FF BELs in Xilinx devices. Dont forget to create an input port cell (with an associated IBUF) for each of the inputs, and an output port cell (with an associated OBUF) for each of the outputs. To verify that the circuit is built correctly, create a dot file of your netlist.

```
// Use your function to create a full adder
CellDesign fullAdder = createFullAdder();

// Print a DOT file of your design
DotFilePrinter printer = new DotFilePrinter(fullAdder);
printer.printNetlistDotFile("fullAdder.dot");
```

To convert the generated DOT file into a viewable image, install GraphViz http://www.graphviz.org/ and run the following command in a terminal:

```
dot -Tpng fullAdder.dot -o fullAdder.png
```

Open the generated fullAdder.png image file, and visually verify that you assembled the full adder correctly, and that the LUTs are correctly initialized.

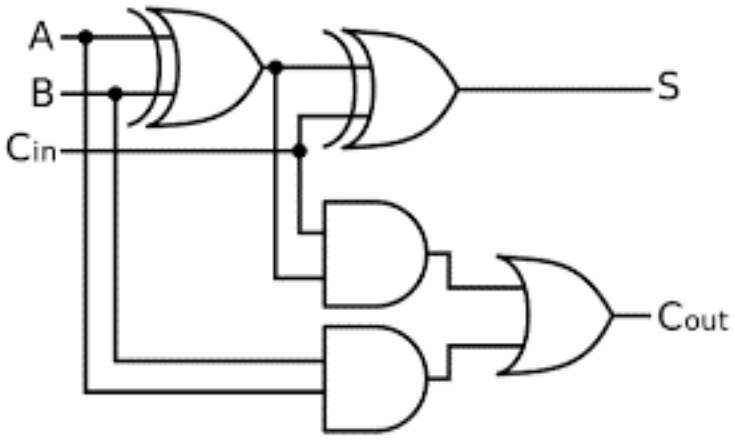

Figure 2: Full Adder Schematic

4 Placing the Full Adder

Using the full adder circuit you created in section 3, place the design onto a Artix7 *xc7a100tcsg324* Xilinx part. This can be done in three different ways:

1. Manually place all cells
2. Randomly place all cells
3. Implement a placement algorithm.

It is suggested that new users start with method 1, and move onto method 2 and 3 as they get more experience with RapidSmith. **NOTE:** placing cells onto BELs is actually a packing problem. Placement is usually done at Site boundaries.

5 Automated Router

Design a simple algorithm to automatically route the INTERSITE portion of a CellNet using RouteTrees. Initially, your algorithm should only route **one sink pin** of the net. Once this works, modify your algorithm to route all sinks of a net. To test your algorithm, load the *cordic.tcp* checkpoint into RapidSmith, route the net named "u2/gen_pipe[8].Pipe/Zo_reg_n_0_[9]", and generate a TCL command that can highlight the used wires in Vivado (shown below).

```
// Load the count16.tcp checkpoint into RapidSmith
String tcpPath = RSEnvironment.defaultEnv()
                  .getEnvironmentPath()
                  .resolve("exampleVivadoDesigns")
                  .resolve("cordic.tcp").toString();

TincrCheckpoint tcp = VivadoInterface.loadTCP(tcpPath);
CellDesign design = tcp.getDesign();
CellNet net = design.getNet("u2/gen_pipe[8].Pipe/Zo_reg_n_0_[9]");

// Call your router
RouteTree route = routeNet(net);

// Print a TCL command to the console for testing
System.out.println(RapidSmithDebug.createHighlightWiresTclCommand(route));
```

Load the equivalent DCP file into Vivado (*exampleVivadoDesigns/cordic.dcp*), and copy the Tcl command generated above into the Tcl console. This will highlight all of the wires that your router used when completing the route. Visually verify that your route starts at the correct source site pin, and ends at the correct sink site pins.