

the pin Z. Notice that the fan-in cone now stops at sequential elements.

Clear the current highlighting and try highlighting a few other cones of logic using `demo_draw_cone` procedure.

## **Summary**

This demonstration walked you through a few Tcl scripts and exposed you to some of the commonly used database access procedures.

With the Tcl scripts you can customize and automate the synthesis flow, perform database access, manipulate the netlist and write utilities for the operations you find yourselves doing frequently.

Overall, BuildGates empowers users with flexibility and advanced programmability through its Tcl interface procedures.

This exercise gave you a flavor of using the database access procedures. All the netlist transversal routines are available whether or not you use schematics and the user interface.

#### **Step 4. Use of Tcl script to draw a fan-in cone.**

NaviGates has a built-in capability of drawing fan-in cones for any pin in the design. Here we demonstrate how a similar utility could be written by the user.

##### **Right-click on the “decode” module in module hierarchy and select “Show Schematic/Main”.**

This will draw the schematic of “decode” module in the main schematic viewer.

##### **Click-select the NOR gate driving the “mem\_wr” port of the design.**

**Make sure you are on the output pin, Z of the NOR gate.**

##### **Right-clicking on the Z pin, select “Fan In Cone, Pin: Z” from the pop-up menu.**

The entire fan-in cone for the pin Z will be highlighted in the schematic. Notice that the colors change with each level of logic. This fan-in cone is traversing through sequential elements in the fan-in cone.

If you wanted to draw a fan-in cone that stops at sequential elements, you could do so using Tcl. We have provided a Tcl script which does just that. Let’s take a look at it.

##### **Click on the “Tcl” tab of schematic viewer.**

##### **Click on “Open” button from the menu bar that shows up. Double-click on the “FaninCone.tcl” file.**

The script implements an algorithm to trace the fan-in cone and highlight it. This particular algorithm uses a depth-first recursive transversal of the input cone.

demo\_draw\_cone procedure takes in a pin id and passes the pin name and instance name to get\_input\_cone procedure. That in turn calls get\_cone\_body procedure with a pin as an argument. Depending on whether it is an input pin or output pin, get\_cone\_body will call get\_cone\_pin\_input or get\_cone\_inst.

get\_cone\_inst is passed a pin. The instance connected to this pin is obtained and highlighted. If the instance is a sequential element, the tracing stops. Otherwise, get\_cone\_pin\_input is called for each input pin of the instance.

get\_cone\_pin\_input takes an input pin. It determines the nets hooked up to that pin, and loops through all pins attached to those nets. If the pin is a port of the containing module, then the port and the net segment attaching the port to the input pin are highlighted. If the pin is an output port, then the net segment is highlighted, and get\_cone\_body is called with the port.

Let’s use this Tcl procedure to draw fan-in cones that stop at sequential elements.

##### **Click on “Save+Source” button from the menu bar.**

Now the procedures defined in FaninCone.tcl are sourced and available to the user.

##### **Click on the “Schematic” tab of schematic viewer.**

##### **Select “Clear” from menus at the top of schematic window to clear the previous highlighting.**

##### **Click-select again the same NOR gate driving the “mem\_wr” port of the design.**

##### **Right-clicking on the Z pin, select “Get Pin ID for Z” from the pop-up menu.**

The pin id for the Z pin is outputted to the ac\_shell command window.

##### **In the ac\_shell command window, type in demo\_draw\_cone <pin\_id> where pin\_id is the id displayed by the previous “Get Pin ID for Z”.**

This will use the newly sourced procedure demo\_draw\_cone and will highlight the fan-in cone for

Let's look at the SetConstraints procedure in detail. It first defines a clock using "set\_clock" command, ties it to a pin of the design using "set\_clock\_arrival\_time", specifies a stiff driver for the clock and then tells BuildGates not to buffer or modify the existing clock tree structure.

We assemble a list of inputs excluding the clock pin and the reset pin. Using the foreach loop, each of the inputs gets a data arrival time and the drive cell constraint. The script then defines the expected time constraints for all the outputs.

Further the script defines the operating condition and wire load mode for the synthesis run.

**Click on "Save+Source" button from the menu bar.**

This will execute the script and now the design has the constraints defined in this script. The script also performs optimization on the design using the constraints.

Thus user can write simple scripts to automate the synthesis flow for BuildGates.

### **Step 3. Use of interactive Tcl commands for the design database access.**

BuildGates provides Tcl procedures to access its database. Some of the frequently used procedures are: get\_names, get\_info, find

Most of database access procedures rely on "object id". All database objects are given an id which TCL uses to refer to those objects. For example, instances, modules, ports and nets are all referenced with an id. Generally, the id of an object is obtained through the use of the find command, or by asking another object for something it's connected to.

get\_names:

The get\_names command takes a list of one or more id's, and returns their names. For instance, if get\_names is passed an id of a net object, then get\_names returns the name of the net.

find:

The find command is a flexible command that returns id's of objects that meet the find criteria passed as it's arguments. Given a name of an object, the find command is used to find the object's id.

get\_info:

The get\_info command is used to find out information about an object. Uses include finding the module name of an instance, getting the net a port is connected to, and getting access to any of an object's properties.

Because TCL is an interpreted programming language, all the commands can be entered directly into the ac\_shell console, and executed immediately by the TCL interpreter.

As an example of interactive TCL used to gather information about the design, let's find out the names of the modules the clock fans out to. We'll build this command step-by-step.

**In the ac\_shell command window, type in** `get_info [find -net clock]`

This will display information about the net "clock"

**Type in** `set pins [get_info [find -net clock] connections]`

We found out the object ids of the connections to "clock" net and assigned a variable "pins" to them.

**Type in**

```
foreach pin $pins {
    puts [get_names [get_info $pin instance]]
}
```

This loops through all the elements of "pins", gets the instance name associated with each of the element and prints it out.

# Advanced Programmability using TCL

This demonstration will familiarize a user to the TCL programming language and will show a few examples of TCL scripts that can help users customize their synthesis environment. We'll also demonstrate certain synthesis database querying techniques useful for circuit manipulation and synthesis flow customization.

The steps followed in the demonstration are

- 1) Read in the RTL of a design.
- 2) Use Tcl script for applying constraints and synthesizing the design.
- 3) Use of interactive Tcl commands for the design database access.
- 4) Use of Tcl script to draw a fan-in cone

## Step 1. Read in the RTL of a design.

We'll read in a design and will show how to write Tcl scripts to customize and automate your synthesis flow.

**Click on the "HDL" icon on the menu bar.**

A "Read HDL files" menu will pop up.

**Select either Verilog or VHDL button at the bottom of the panel.**

The HDL files for that language will appear in the "Files" list.

**Use the left button to drag over the entire list of HDL files in the "Files" list. Keeping the pointer in the selected files pane, click the middle mouse button.**

This transfers the files to the lower file list.

**Click on the "OK" button at the bottom of the "Read HDL files" panel.**

This step will read in all the RTL files into BuildGates.

**From the main menu bar, click-select the "Tools/Build Generic" menu item.**

This step will first link the whole design, will report the case statement statistics and register inferences found in the RTL. The design is now synthesized into a technology independent netlist form. The hierarchy tree of the design will pop-up in the Module Hierarchy Viewer.

## Step 2. Use Tcl script for applying constraints and synthesizing the design.

Now that we've read in a design, let's look at a Tcl script that will apply synthesis constraints on the design and synthesize it.

BuildGates does not require complex scripting just to perform simple synthesis. You can synthesize a design interactively in NaviGates or write a simple command file to run BuildGates in batch mode. Advanced Tcl scripting is a way of automating and customizing synthesis flow for your needs.

**Click on the "Tcl" tab of schematic viewer.**

**Click on "Open" button from the menu bar that shows up. Double-click on the "build.tcl" file.**

The build.tcl file appears in the window. The SetConstraints procedure applies constraints to the design. The "if-else" structure towards the end of the file, reads in the lca300k technology library, applies constraints to the design using the SetConstraints procedure and then synthesizes the design.