Exercise18 (Synthesizing balance_cntrl):

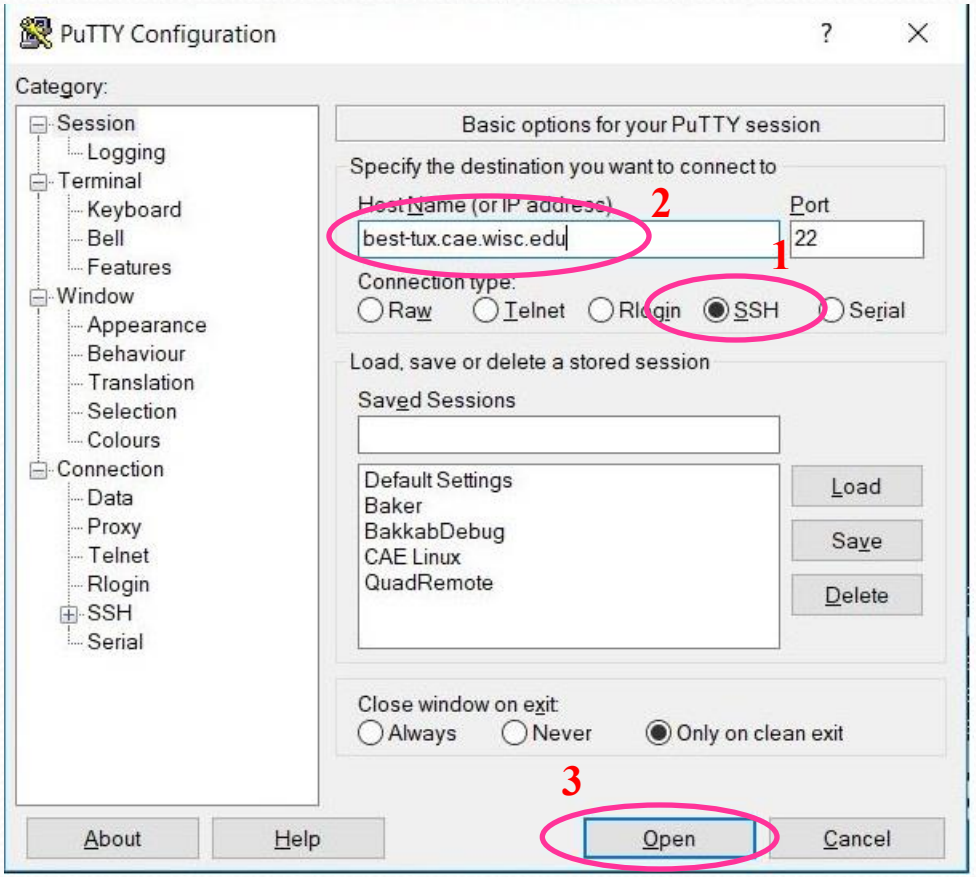
- Synthesizing ***balance_cntrl*** *(which has children **PID** & **SegwayMath**)*
- We may run out of Synopsys licenses if we all try to get on
  - Work with a partner
  - **Both** submit the script and results you get by end of class
- Synopsys only works on Linux

Exercise18 (Synthesizing balance_cntrl):

- You have coded and validated **balance_cntrl** and its children (PID & SegwayMath) now it is time to synthesize them.

- Using PuTTy (installed on CAE machines) login to a Linux machine (**easy as 1,2,3**)

- Once logged in to Linux do an **ls**. You should see you have an **ece551** directory already (at least you should if you created on in your **I:drive** area.) Everything on your **I:drive** exists on Linux…magic!



Exercise18 (Synthesizing balance_cntrl):

- Since your I:drive is mapped to your linux home and vice versa file transfer is not necessary. Create an exercise18 directory under your ece551 directory.
  - You can do this in linux by doing a: **cd ece551** and then doing a: **mkdir exercise18**
  - OR you can create it in Windows and it will appear in your Linux.

- Move your **balance_cntrl.sv**, **PID.sv,** and **SegwayMath.sv** files to your exercise18 directory.

- Now we will kick off Synopsys in command shell mode (which is how "real" engineers use it)

  - **Unix_prompt> design_vision –shell dc_shell**

Exercise18 (Synthesizing balance_cntrl):

- Write a synthesis script (**balance_cntrl.dc**). The script should perform the following:
  - Defines a clock of 125MHz frequency and sources it to clock
  - Performs a set don't touch on the clock network
  - Defines input delays of 0.3 ns on all inputs other than clock
  - Defines a drive strength equivalent to a 2-input nand of size 2 from the Synopsys 32nm library (NAND2X2_RVT) for all inputs except clk and rst_n
  - Defines an output delay of 0.75ns on all outputs.
  - Defines a 50fF load on all outputs.
  - Sets a max transition time of 0.15ns on all nodes.
  - Employs the Synopsys 32nm wire load model for a block of size 16000 sq microns
  - Compiles, then flattens the design so it has no hierarchy, and compiles again.
  - Produces a min_delay report
  - Produces a max_delay report
  - Produces an area report
  - Flattens the design so it has no hierarchy
  - Writes out the gate level verilog netlist (**balance_cntrl.vg**)

NOTE: Library default units are pF not fF

- Submit to the dropbox for Exercise18
  - Your synthesis script (**balance_cntrl.dc**)
  - The output reports for area (**area.txt**)
  - The gate level verilog netlist (**balance_cntrl.vg**)

For reference my area was 3250ish square microns

Hints for Synthesis Scripting.

- About 1/3 of the way through Lecture06 there is an example synthesis script.
- A synthesis script is just a series of commands that you can directly type into ***dc_shell***. Always test each command in ***dc_shell*** first to make sure you are using it right, then copy that line into your synthesis script.
- When reading in System Verilog you have to use: **read_file –format sverilog {file_names}**. Note sverilog not verilog.
- When reading in several files of a design that have hierarchy it is best to do as follows:

```
read_file -format sverilog {UART_tx.sv UART_rcv.sv UART.v}
```

  - NOTE: reading the children first, and parents later

- Then it is important to set the level you wish to synthesize next

```
set current_design UART
```

- Also can be necessary to get it to traverse the design hierarchy so it knows who the children are:

```
link
```