Homework 1

Gate-level design and simulation using TkGate

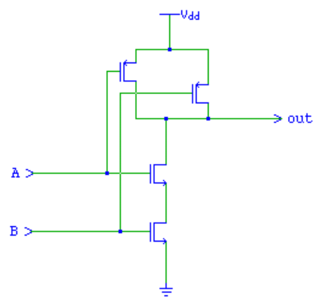
A. Tutorial

Take the first 7 steps of the TkGate tutorial provided with the tool:

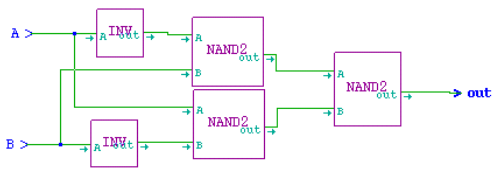
1. TkGate Introduction
2. Creating a Circuit
3. Basic Editing Modes
4. Group Editing Features
5. Editing wires
   Wires may represent single wires, carrying 1 bit at the time, and multi-bit channels. The bit-width of a wire is the number of parallel wires it represents. The tutorial doesn't explain how to connect two wires with different bit widths. This is especially useful to group several wires to form a BUS, as illustrated in section B. The bits of a n-bit wire are indexed from 0 to n-1. When you connect a thin wire to a wider one, small numbers appear at the intersection to represent the "bit range", that is the set of bits of the larger wire that have been actually connected to the smaller one. You can edit the Bit range by double-clicking at the intersection of the two wires, and opening the "Details" folder of the popup window.
6. Using Modules
   Each module is composed of an interface and an implementation. You need to create the interface first, and then specify the implementation. The tutorial explains how to create an both the interface and the implementation, but it doesn't explain how to link them together. Once the interface has been created, the "Module->Set interface" command has to be issues in order to make it effective. When you enter the module, the names of its I/O ports appears on the bottom-left corner of the window. You need to add to the implementation input and output elements with the same names in order to link internal wires to the interface. This can be done by using the "Make->Module->Module input" and "Make->Module->Module output" commands.
7. Combinational Simulation

B. Hierarchical design and simulation of a 8-bit parity checker

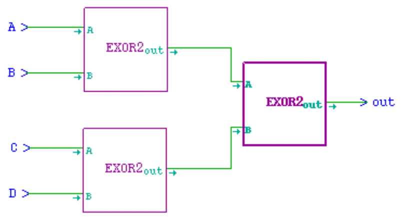
1. Create a 1-input 1-output module called INV, enter the module and provide a CMOS implementation of the inverter. Notice that **nMOS** and **pMOS** transistors are available in the Make->Gate menu, while power supply lines **Ground** and **Vdd** (representing respectively constant logic values "0" and "1") are available in the Make->I/O menu.
2. Create a 2-input, 1-output module called NAND, enter the module and provide a CMOS implementation of the NAND.



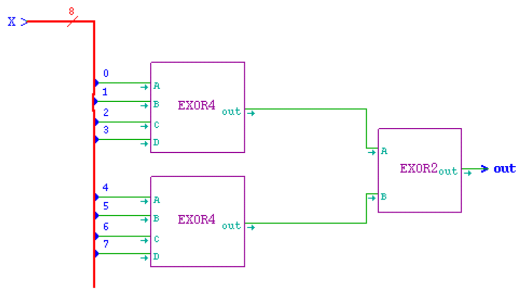
3. Create a 2-input 1-output module called EXOR2, enter the module and provide a NAND-based implementation of the exor using the NAND and INV modules created so far.



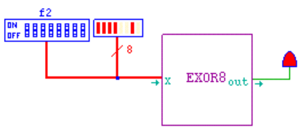
4. Create a 4-input 1-output module called EXOR4, enter the module and implement the 4-bit exor using 3 instances of the EXOR2 module.



5. Create a module called EXOR8, with a 8-bit input port and a 1-bit output port. Enter the module and implement a 8-bit exor using the EXOR2 and EXOR4 modules created so far.



6. At the top level connect the input of the EXOR8 module to a 8-bit deep-switch, and the output port to a led.



7. Run the simulation and test the circuit: the led must be red whenever the input configuration contains an odd number of 1's. Notice that the input configuration is specified by means of 2 hexadecimal digits defined on the alphabet 0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f.

You can compare your design with the solution provided in file **exor.v**.