This file can be found in /comp/150CAD/public_html/HWs/gmf/network_reader.pdf .
The C++ library files discussed below can be found in /comp/150CAD/public_html/HWs/code .

You will use a network reader as a library for most of the programming assignments in this course. It comes in two files:

- gmf_parser.cxx has routines to read a text file containing a network description, parse them and call action routines from gmf_build_network.cxx. The network format is extremely simple; it is meant to keep things easy and let us focus on bigger issues.
- gmf_build_network.cxx has routines that are called from gmf_parser.cxx, and which build network data structures. It also has routines to print them (for debug purposes) and has several interesting functions that can help you with your homeworks.
- gmf.hxx contains the function declarations to make the above two files work. You should include it in your own files that use the network-reader library.

The external variables and data structures from gmf.hxx are:

- enum OpType: the types of cells in our library: e.g., NAND, LATCH. We only have simple ones!
- Node: a structure that describes one node. Mostly, it holds the OpType of the gate driving this node, as well as that gate's inputs. It also holds the node's fanout nodes and a few other miscellaneous fields.
- vector<Node> g_nodes: all of the nodes in the network. This gets built for you by the network reader.
- typedef int NodePtr: a node pointer is really just an integer index into g_nodes, but we give it its own type to aid in clarity.

The following functions will be useful:

- parse_gmf (string filename): read a network-description file and build g_nodes[].
- void printNodes (): print out the entire network (e.g., for debugging).
- void gen_one_gate(NodePtr np): for use in the levelized-compiled-code homework. It prints (to the standard output) the code for the one gate *np*. So if, e.g., *np* is the AND gate with output Q and inputs A, B and C, it would print "Q = A & B & C". It does not work for flops.
- bool op (NodePtr np): for use in the event-driven simulation homework. It looks at the value of the node *np*'s inputs and computes the value of *np*. E.g., with the same AND gate as above, if we had A=1, B=1 and C=0 then it would return false. It assumes that you store each node's current value in its g_nodes[np].value field.
- int n_input_phases(): in addition to simply describing the network, the network-description file also assigns each primary input a sequence of values in the Node.input_data vector. For simulation, this typically holds the excitation values to drive the network. For timing analysis, it holds primary-input arrival time. The function n_input_phases() checks to ensure that all PIs have the same number of values given, and returns that number.