Dry 2:

1. We used 2 “queue” data structures to hold the elements in the netlist.

By using HCM properties we added a “Value” field to the “hcmNode” structure and “Visited” field to the “hcmInstance” structure.

Every time that node’s value gets updated, the logical gate that is connected to that value through “IN” port is inserted to “gate\_queue”. In a same way, when a logical gate’s output is updated, the node that is connected to the relevant “OUT” port is inserted to the “node\_queue”.

\*Within the queue there is no priority on which element should be dealt first, since all of the events (or gates) that are held in a the same queue are modeled to happen simultaneously.

1. The algorithm of schedule control works in a follow way:
2. Applying input vectors on input nodes (as “Value” parameter).
3. Push to “node\_queue” all the nodes that signal’s value got changed.
4. Pop all the elements of “node\_queue” one by one, for each element (node) find all instances that are connected to this node through “IN” port and add those instances to the “gate\_queue”. By end of this step “node\_queue” is supposed to be empty.
5. Pop all the elements of “gate\_queue” one by one, for each element (instance) perform logical gate evaluation and if the output signal of that instance got changed, push a node that is connected to the instance’s “OUT” port to the “node\_queue”.

By the end of this step “gate\_queue” is supposed to be empty.

1. If “node\_queue" is not empty go to “c” step, else take next input vector, increase the time and jump to “a” step. If it was the last input vector – finished.

The algorithm is looped in 2 loops, the internal loop will break only if all the signals are stabilized, hence both queues are empty. The external loop will follow until the program reads all the input vectors.

Pseudo-code: ( for simplification, in pseudo-code we are assuming that each element in queue is unique, in actual code we are implementing it)

while (current\_vector != NULL ) {

for (node in flat\_cell -> GetNodes){

if( node->top\_cell\_port == IN){

if (node->value != current\_vector){

node->value = current\_vector;

node\_queue.push(node);

}

}

}

while (!node\_queue.empty){

curNode = node\_queue.pop;

for ( instPort in curNode ->GetInstPorts){

if(InstPort == IN){

gate\_queue.push(InstPort->Instance)

{

{

while(!gate\_queue.empty){

curInst = gate\_queue.pop;

for (instPort in curInst->GetInstPorts){

if (InstPort == OUT){

old\_value = InstPort->Node->value;

evaluate\_gate(curInst);

cur\_value = InstPort->Node->value;

if (old\_value != cur\_value){

old\_value = cur\_value;

node\_queue.push(InstPort->Node);

}

}

}

}

}

1. In our model we are assuming that the time is progressing by one time unit each time new vector is applied, meaning that internal logic (combinatorial and procedural) is based on zero delay mode. Therefore there is no need to model any time within a single cycle.
2. We can only simulate cells that are included in “stdcell.v” or are built from them.

We cant have loops in a simulation, meaning that output signal goes back to input, this will lead to the endless loop simulation.