Wet5 – Documentation:

1. The high level algorithm is as follows:
   1. Flatten each of the input circuits
   2. Initiate a circuit inputs map (PI\_map) and circuit outputs map (PO\_map). Each pair in the maps holds a node from the first input circuit and its corresponding node from the second input.
   3. For each circuit, find all FFs. Add their output nodes to PI\_map and their input nodes to PO\_map. Set each FF output/input an identifying integer (“sat var”). Each node in the PI\_map also gets a “PI” property that identifies it as a PI.
   4. Repeat the process for all top inputs and outputs of the circuits. If the name of an input/output node from the first circuit cannot be found in the second one, exit (as we assume the top input/outputs match).
   5. Assign an integer identifier for all the nodes left unassigned in both circuits, i.e. inner nodes that are not connected to FFs. These do not have to match between circuits.
   6. For each node in a PO set, traverse the circuit and calculate its clauses via recursion. Save calculated clauses for each node to eliminate repetitive calculations (in the next PO). If we find an undriven node we print an error message and exit the program. Repeat for all PO sets.
   7. Once all the nodes in PO\_map are assigned their accumulative clauses, compare between the nodes in each pair using XOR gate and SAT solver. If they are not equal, print out their names. If all pairs are equal, the circuits are equal.
2. As described, for each pair in the PO\_map, for each node’s path we first calculate the accumulative set of clauses. Then, we calculate a XOR clause, with the nodes’ identifiers as its inputs. Finally, we concatenate all the clauses to one vector and activate the SAT solver. Since we used a XOR gate, if the SAT succeeded in solving the DIMACs then the circuits are necessarily not logically equally. If for all output pairs the SAT solver failed to find a solution, then the two circuits are equal.
3. In order to handle constants we add a parameter “constant” to a node that can be either 0 or 1 based on the constant value. When we are calculating the clauses for a gate we first check to see if any of the input nodes are constant and based on that we alter the output. For instance if there is a constant 1 at the input to an OR gate we add the parameter “constant to the output node with the value 1 and return the clause of the output sat variable. There are also cases where an input constant can affect the gate clauses without causing the output to be constant.
4. Since the clause expression for XOR growth nonlinearly for the number if inputs, we decided to support only XOR2 gates. This is per the content of stdcell.v, which does not include XOR gates with more than two inputs. The XOR clauses are: where ‘out’ is the XOR output and ‘in1’, ‘in2’ are its inputs.