# Logic Synthesis & Verification, Fall 2024

**National Taiwan University**

## Programming Assignment 2

Due on 12/1 23:59 on GitHub.

*Submission Guidelines.* Please develop your code under `src/ext-lsv`. Please do not modify any code outside `src/ext-lsv`, and we will only copy your files under `src/ext-lsv` for evaluation. You are asked to submit your assignments by creating pull requests to your own branch. To avoid plagiarism, please push files and create pull requests after 9 p.m. on the due day. Please see the GitHub page (https://github.com/NTU-ALComLab/LSV-PA) for more details.

## 1 [Satisfiability Don't Cares Computation] (50%)

### 1.1 Problem Description

Please implement command "`lsv_sdc`" in ABC (under `src/ext-lsv/`), so that after reading in a circuit (by command "`read`") and transforming it into AIG (by command "`strash`"), running the command "`lsv_sdc`" would invoke your code and compute the satisfiability don't cares (SDCs) on the fanins of a given node $n$.

The SDCs of $n$ are the variable assignments of its fanin nodes $Y$ that do not appear under any primary input assignments, as given by

$$SDC_n(Y) = \forall X. \bigvee_{y \in Y} (y \oplus F_y(X)), \tag{1}$$

where $X$ is the set of primary input variables and $F_y(X)$ is the function of fanin node $y \in Y$ in terms of $X$.

### 1.2 Commmand Format

The command should have the following input format:

`lsv_sdc <`$n$`>`

where $n$ is an internal node (excluding constant, PI, PO nodes). Your command should compute the SDCs in terms of the fanins of the given node $n$.

For the terminal output, please list all the minterms of the satisfiability dont'cares. Please use 0 and 1 to express the minterms following the order of fanin variables. Note that you should take the fanin negation into account. If there are no SDCs, please output "`no sdc`".

For example, given the network in Fig. 1. Some example outputs are as follows:

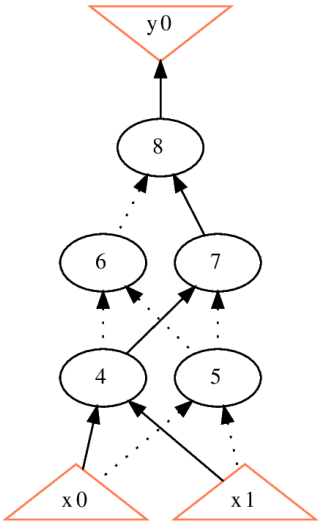


**Fig. 1.** A simple network to demonstrate SDC and ODC.

```
abc 01> lsv_sdc 7
10
abc 02> lsv_sdc 6
00
abc 02> lsv_sdc 5
no sdc
```

### 1.3 Implementation Guidelines

Please implement the command using random simulation and SAT solving as follows.

1. Perform random simulation and observe the values on the fanin nodes $y_0, y_1$ of $n$.
2. For those patterns (among 00, 01, 10, 11) that do not appear on $y_0, y_1$, use SAT solver to check whether the pattern can appear on $y_0, y_1$.

To check whether a pattern (value assignment) $(v_0, v_1)$ can appears on $y_0, y_1$:

1. Build a CNF formula from the subcircuit consists of the transitive fanin cones of $y_0$ and $y_1$ using Tseytin transformation.
2. Assume the value of variables $y_0$ and $y_1$ to be $v_0$ and $v_1$ respectively.
3. Solve the CNF formulation. If SAT, then $(v_0, v_1)$ is not an SDC. Else, $(v_0, v_1)$ is an SDC of $n$ in terms of $y_0, y_1$.

## 2 [Observability Don't Cares Computation] (50%)

### 2.1 Problem Description

Please implement command "`lsv_odc`" in ABC (under `src/ext-lsv/`), so that after reading in a circuit (by command "`read`") and transforming it into AIG (by command "`strash`"), running the command "`lsv_odc`" would invoke your code and compute the local observability don't cares (ODCs) of an AIG node $n$ with respect to all primary outputs $Z$ in terms of its fanin nodes $Y$.

The local ODCs are the fanin variable assignments under which the value of $n$ does not affect the output value of any of the primary outputs.

For more details, please refered to lecture 07 page 11 to 29.

### 2.2 Command Format

The command should have the following input format.

`lsv_odc <`$n$`>`

Your command should compute the local observability don't cares of $n$ with respect to all primary outputs $Z$ in terms of its fanin nodes $Y$.

For the terminal output, please list all the minterms of the observability don't cares. Please use 0 and 1 to express the minterms following the variable order of primary inputs. Note that you should take the fanin negation into account. If there are no ODCs, please output "`no odc`"

For example, given the AIG in Fig. 1 Some commands and the corresponding outputs are as follows:

```
abc 01> lsv_odc 7
01
abc 02> lsv_odc 8
no odc
```

Note that `10` is SDC but not ODC for node `7`.

### 2.3 Implementation Guidelines

Please implement the command using SAT solving as follows.

1. Let $C$ be the given AIG.
2. Let $C'$ be the same as $C$ except that node $n$ is negated.
3. Build a CNF formula of the miter of $C$ and $C'$. The miter of two designs, as shown in fig. 2, is a circuit that output 1 if and only if any pair of the outputs from these two designs are different under the same input value. More specifically, the two designs share the same primary inputs, and their outputs are pairwise XOR and then feed into an OR gate, which would be the output of the miter.

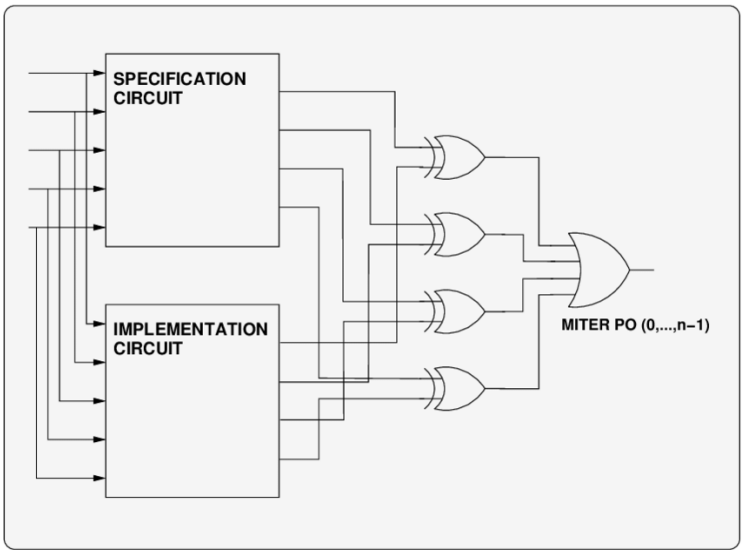


**Fig. 2.** A miter of two designs (specification and implementation circuits.)

4. Assume the miter output is 1 and solve ALLSAT to obtain the set of satisfying assignments on $y_0, y_1$. These assignments would be the care set of $n$ in terms of $y_0, y_1$.
5. The assigments that are not in the care set are the don't cares of $n$ in terms of $y_0, y_1$. This includes both SDCs and ODCs. You should remove SDCs computed in part 1 and report only ODCs.

To solve ALLSAT and obtain all the satisfying assignments on variables $y_0$ and $y_1$, after each SAT solving, you can block the assignment $(v_0, v_1)$ you found by adding a caluse

$$(y_0 \oplus v_0 \vee y_1 \oplus v_1)$$

and solve the new CNF formula again. The new satisfying solution will have different value assignment on $y_0$ and $y_1$.

## 3 References

Hint 1. Here are some useful functions when performing SAT solving (and creating miter circuits)
(1) Use `Abc_NtkCreateCone` to extract the cone of $y_k$.
(2) Use `Abc_NtkToDar` to derive a corresponding AIG circuit.
(3) Use `sat_solver_new` to initialize an SAT solver.
(4) Use `Cnf_Derive` to obtain the corresponding CNF formula $C_A$, which depends on variables $v_1, \ldots, v_n$.

(5) Use `Cnf_DataWriteIntoSolverInt` to add the CNF into the SAT solver.
(6) Use `Cnf_DataLift` to create another CNF formula $C_B$ that depends on different input variables $v_{n+1}, \ldots, v_{2n}$. Again, add the CNF into the SAT solver.
(7) For each input $x_t$ of the circuit, find its corresponding CNF variables $v_A(t)$ in $C_A$ and $v_B(t)$ in $C_B$. Set $v_A(t) = v_B(t) \forall t \notin \{i, j\}$, and set $v_A(i) = v_B(j)$, $v_A(j) = v_B(i)$. This step can be done by adding corresponding clauses to the SAT solver.
(8) Use `sat_solver_solve` to solve the SAT formula.
(9) Use `sat_solver_var_value` to obtain the satisfying assignment.

Hint 2. To use `Abc_NtkToDar` and `Cnf_Derive` functions, you should include the following code.

```
#include "sat/cnf/cnf.h"
extern "C"{
    Aig_Man_t* Abc_NtkToDar( Abc_Ntk_t * pNtk, int fExors, int fRegisters );
}
```

Hint 3. The variable orders in CNF differ from the ones in AIG. For a pointer `pObj` in `Aig_Obj_t*` type, you can use `pCnf->pVarNums[pObj->ID]` to find its variable index in `pCnf`. If the pointer `pObj` is from the original network in `Abc_Obj_t*` type, to make it have the same ID as its counterpart in AIG, make sure you set the last parameter of `Abc_NtkCreateCone` to 1 in step (1), so that all input variables are always included.

Hint 4. You can refer to our GitHub page (https://github.com/NTU-ALComLab/LSV-PA/wiki/Reasoning-with-SAT-solvers) for more details about using SAT solvers in ABC.