Department of Electrical Engineering
National Taiwan University
September 17, 2025

Introduction to EDA, Fall 2025

Programming Assignment #1: Equivalence Checking

Due by October 8, 2025, 23:59

1 Introduction

Equivalence checking is an essential task in functional verification. In this assignment, you will study the problem of verifying whether two combinational circuits are equivalent. The equivalence checking (EC) problem can be formulated as a conjunctive normal form (CNF) and solved using a SAT solver.

Specifically, you will generate a CNF representation of a miter circuit composed of two input circuits, and then use `MiniSat` to determine whether the circuits are functionally equivalent.

2 Preliminaries

2.1 DIMACS CNF Format

The DIMACS format is the standard input format for most SAT solvers. A CNF file in DIMACS format consists of:

- A problem line starting with `p cnf`, followed by the number of variables and the number of clauses. Example: `p cnf 12 27`
- Each clause is a line of integers terminated by a `0`. Positive integers represent variables, while negative integers represent their negations. Example: `1 -5 0` means $(x_1 \vee \neg x_5)$.

2.2 MiniSat

MiniSat is a popular SAT solver that takes a DIMACS CNF file as input. It outputs either:

- `SATISFIABLE` with a satisfying assignment
- `UNSATISFIABLE`

You may install MiniSat from `http://minisat.se/MiniSat.html` and run it as:

```
./MiniSat_v1.14_linux [*.dimacs] [*.output]
```

3 Problem Statement

3.1 Brief Description

You are required to write a program that generates the CNF for verifying the equivalence of two combinational circuits. A **miter structure** is introduced to compare circuit outputs. Two circuits C_1 and C_2 are equivalent if and only if the miter's output is always constant 0.

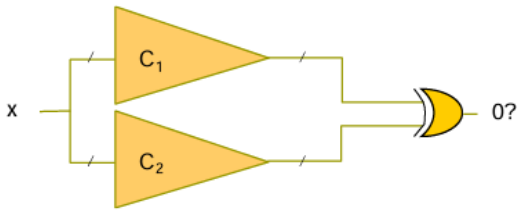

Figure 1: Miter structure for two combinational circuits.

Each combinational gate should be translated into CNF clauses using Tseitin transformation. You can use clauses with single variable to handle the final output of the whole miter circuits.

Type	Operation	CNF sub-expression
AND	$C = A \cdot B$	$(\overline{A} \vee \overline{B} \vee C) \wedge (A \vee \overline{C}) \wedge (B \vee \overline{C})$
NAND	$C = \overline{A \cdot B}$	$(\overline{A} \vee \overline{B} \vee \overline{C}) \wedge (A \vee C) \wedge (B \vee C)$
OR	$C = A + B$	$(A \vee B \vee \overline{C}) \wedge (\overline{A} \vee C) \wedge (\overline{B} \vee C)$
NOR	$C = \overline{A + B}$	$(A \vee B \vee C) \wedge (\overline{A} \vee \overline{C}) \wedge (\overline{B} \vee \overline{C})$
NOT	$C = \overline{A}$	$(\overline{A} \vee \overline{C}) \wedge (A \vee C)$
XOR	$C = A \oplus B$	$(\overline{A} \vee \overline{B} \vee \overline{C}) \wedge (A \vee B \vee \overline{C}) \wedge (A \vee \overline{B} \vee C) \wedge (\overline{A} \vee B \vee C)$

Figure 2: Tseitin transformation.

3.2 Input/Output Specification

Your program binary should be named `ec`. The program should be invoked as:

```
./ec [*_A.bench] [*_B.bench] [*.dimacs]
```

Input: Two bench files describing the combinational circuits. Each consists of:

- A list of primary inputs declared by `INPUT(...)`.
- A list of primary outputs declared by `OUTPUT(...)`.
- A set of gate definitions in the form:

```
out_pin = GATE(in_pin1, in_pin2, ...)
```

Here are some important notes:

- The supported gate types are: `AND`, `NAND`, `OR`, `NOR`, `NOT`, `XOR`, `BUFF`.
- Gates `AND`, `NAND`, `OR`, and `NOR` may have **more than two inputs**.
- Pin names are **not restricted to numbers** (e.g., `a`, `b2`, `net_45`).

Output: A DIMACS CNF file representing the miter circuit.

The CNF will be checked by MiniSat. If satisfiable, the circuits are **not equivalent**; otherwise, they are **equivalent**.

3.3 Example

Sample Input: Circuit A and Circuit B as shown below.

Circuit A:

```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)

OUTPUT(5)
OUTPUT(6)

7 = OR(2, 3)
5 = AND(1, 7)
6 = AND(4, 7)
```

Circuit B:

```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)

OUTPUT(5)
OUTPUT(6)

5 = AND(1, 2)
6 = AND(3, 4)
```

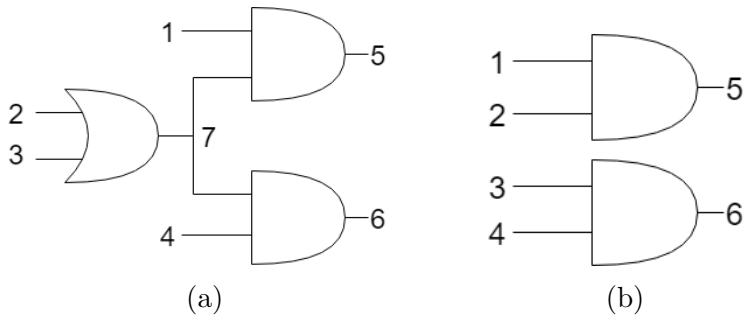

Figure 3: Example input combinational circuits. (a) Circuit A, (b) Circuit B.

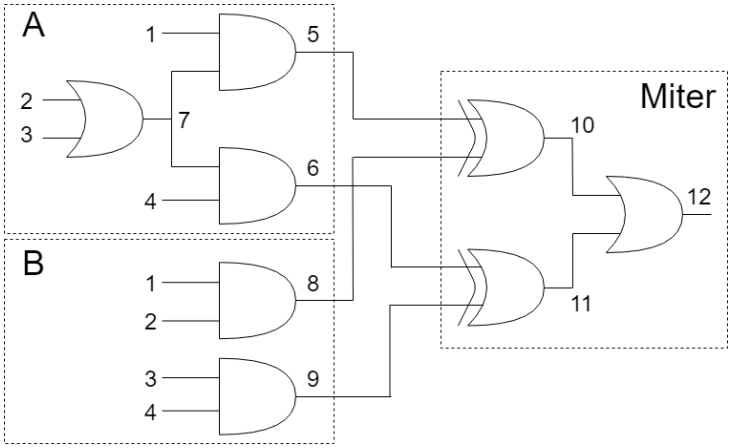

Figure 4: The miter structure generated from Circuit A and B.

Sample Output: A CNF file in DIMACS format, e.g.:

```
p cnf 12 27
1 -5 0
7 -5 0
5 -1 -7 0
...
```

```
IRISLAB-79:~/IntroToEDA/PA1> ./MiniSat_v1.14_linux example.dimacs example.output
============================[MINISAT]===================================
| Conflicts |     ORIGINAL     |              LEARNT              | Progress |
|           | Clauses Literals |   Limit Clauses Literals  Lit/Cl |          |
==========================================================================
|         0 |       24       66 |       8       0       0     nan |  0.000 % |
==========================================================================
restarts              : 1
conflicts             : 0              (0 /sec)
decisions             : 3              (3058 /sec)
propagations          : 12             (12232 /sec)
conflict literals     : 0              ( nan % deleted)
Memory used           : 1.69 MB
CPU time              : 0.000981 s

SATISFIABLE
```

Figure 5: Snapshot of MiniSat log for the example CNF.

4 Evaluation

- Your program must handle circuits with thousands of gates.
- We will use `MiniSat` to check the CNF files generated by your program.
- Correctness is the primary evaluation criterion.
- If your program takes more than 1 minute to generate CNF, the case will be considered failed, even if the result is correct.

5 Submission

- **Plagiarism must be avoided.** All submissions will be checked for similarity. If plagiarism is detected, the grade will be divided among all students involved.
- Compress your project directory into a single zip file named: `StudentID_pa1.zip` (e.g., `b09901001_pa1.zip`).
- The unzipped directory must follow the structure below:

```
<StudentID_pa1>/          # e.g., b09901001_pa1
 |
 |-- bin/                 # compiled binary (e.g., ec)
 |-- src/                 # source code (e.g., main.cpp)
 |-- Makefile
 |-- README.md
```

- The binary executable should be located in the `bin/` directory.
- The source files should be in `src/`.
- The makefile should generate the binary into `bin/`.
- The README must clearly describe compilation and execution instructions.
- **Penalty for late submission: 20% per day. (Deadline: October 8, 2025, 23:59.)**