PROJECT 4:

Nikolay Nikolov

Problem 1:

Design a combinational circuit C1 with the following requirements.

The combinational logic C1 should contain at least 8 gates which include at least 3 OR and 3 NAND gates. Make sure that your design be different from those used by other students.

If your design is identical to the one used by other students, further investigation will be conducted and you will be asked to revise your design and redo the work.

Task 1:

1) Draw your circuit

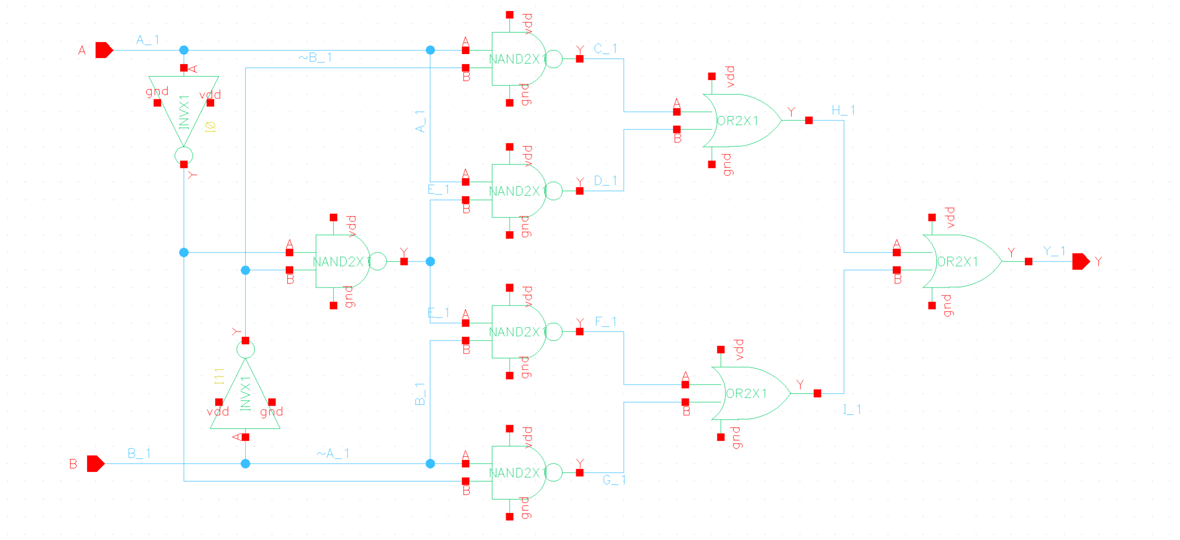


Fig. 1: Circuit Diagram for C1

2) You make an identical copy of C1 and name it as C2. Except the inputs, add the index 1 to all the wire names of C1, while adding the index 2 to all the wire names of C2.
Assume that the corresponding inputs use the same variable names, respectively.

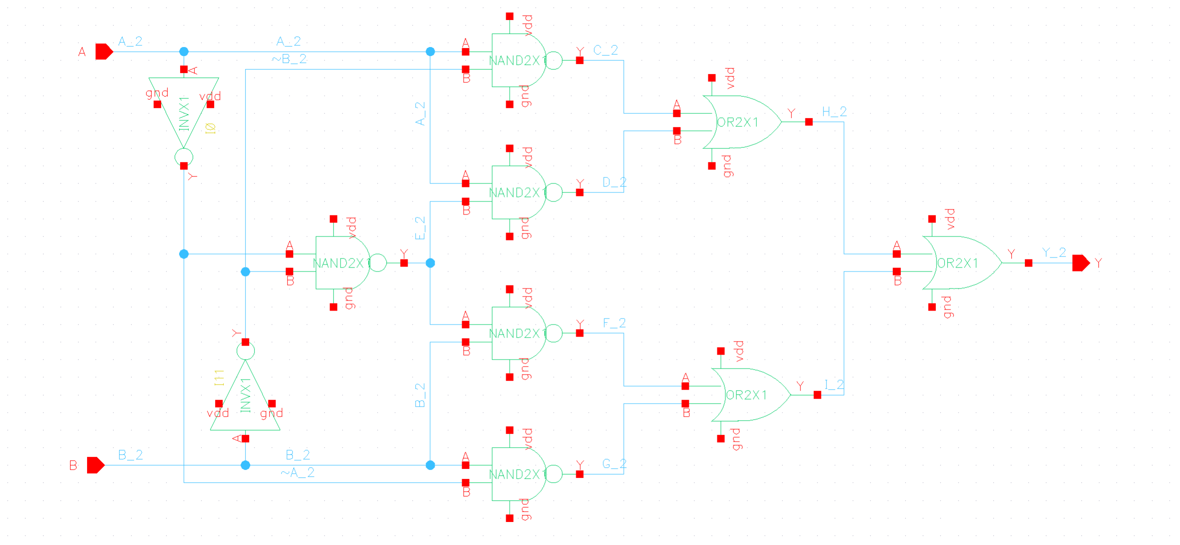


Fig. 2: C2 diagram

3) Download a SAT solver for satisfiability.

4) Prove the equivalence of C1 and C2 by a SAT solver.

```
c Nikolay Nikolov
c Mini−SAT ECE582
p cnf 10 8
−1 −2 5 0
2 −1 7 0
2 5 6 0
1 5 4 0
1 −2 3 0
3 4 8 0
6 7 9 0
8 9 10 0

WARNING: for repeatability , setting FPU to use double precision
============================[ Problem Statistics
    ]=============================
|

    |
|  Number of variables:            10
                                              |
|  Number of clauses:               8
                                              |
|  Parse time:                   0.00 s
                                            |
|  Eliminated clauses:           0.00 Mb
                                           |
|  Simplification time:          0.00 s
                                            |
|

    |
============================[ Search Statistics
    ]==============================
| Conflicts |            ORIGINAL            |            LEARNT
    | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl
    |           |
===============================================================================

===============================================================================

restarts              : 1
conflicts             : 0               (0 /sec)
decisions             : 1               (0.00 % random) (1230 /sec)
propagations          : 0               (0 /sec)
conflict literals     : 0               (−nan % deleted)
Memory used           : 12.00 MB
CPU time              : 0.000813 s

SATISFIABLE
SAT −1 −2 −3 4 5 −6 7 8 9 −10 0
```

Task 2:

Get a circuit C3 from C2 by replacing one gate in C2 with a gate of different functionality. Prove or disprove C1=C3 by a SAT solver.

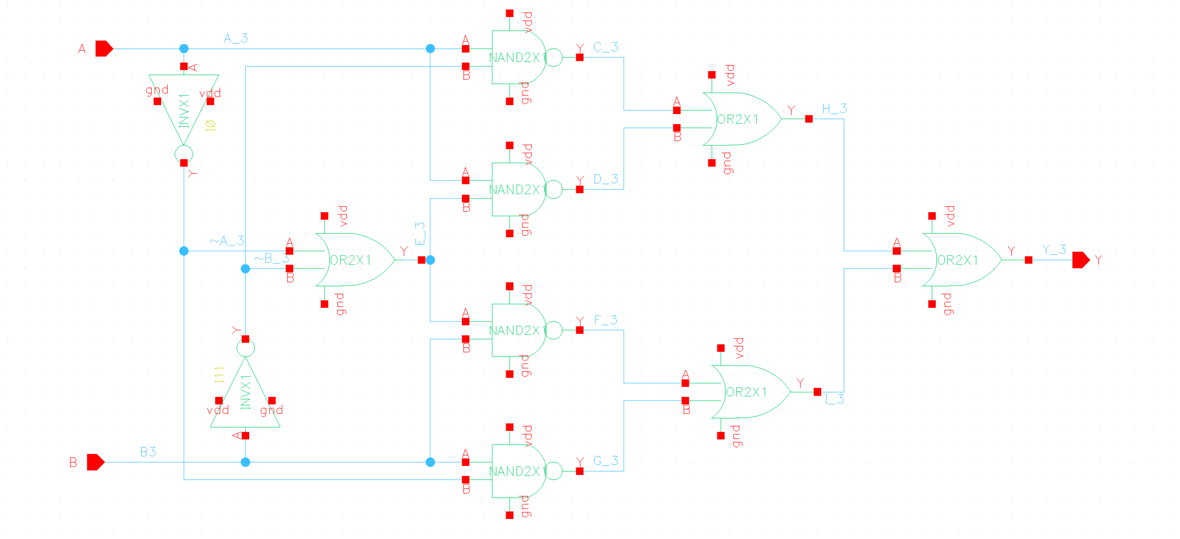


Fig. 3: C3 Circuit diagram, changing NAND gate at E_3 with OR gate

```
WARNING:  for  repeatability ,  setting  FPU  to  use  double  precision
============================[ Problem  Statistics
    ]============================
|

    |
|   Number  of  variables:              10
                                                |
|   Number  of  clauses:                 8
                                                |
|   Parse  time:                      0.00  s
                                              |
|   Eliminated  clauses:              0.00  Mb
                                             |
|   Simplification  time:             0.00  s
                                              |
|

    |
============================[ Search  Statistics
    ]=============================
|  Conflicts  |            ORIGINAL            |            LEARNT
    |  Progress  |
|             |    Vars   Clauses  Literals  |     Limit    Clauses  Lit/Cl
    |            |
================================================================================
================================================================================
restarts              : 1
conflicts             : 0              (-nan /sec)
decisions             : 1              (0.00 % random) (inf /sec)
propagations          : 0              (-nan /sec)
conflict literals     : 0              (-nan % deleted)
Memory used           : 12.00 MB
CPU time              : 0 s

SATISFIABLE

SAT -1 -2 -3 4 5 -6 7 8 9 -10 0
```

All circuits have the same SAT. Hence they are equivalent. Adding the SAT in the input also gave a satisfiable solution.

```
c Nikolay Nikolov
c Mini-SAT ECE582
p cnf 10 8
1 2 5 0
2 -1 7 0
2 5 6 0
1 5 4 0
1 -2 3 0
3 4 8 0
6 7 9 0
8 9 10 0
-1 -2 -3 4 5 -6 7 8 9 -10 0

nik@nik~/Downloads/minisat/build $ ./minisat-simp minisat.in
    minisat.out
WARNING: for repeatability, setting FPU to use double precision
============================[ Problem Statistics
    ]=============================
|

    |
WARNING! DIMACS header mismatch: wrong number of clauses.
|  Number of variables:           10
                                          |
|  Number of clauses:              9
                                          |
|  Parse time:                  0.00 s
                                        |
|  Eliminated clauses:          0.00 Mb
                                       |
|  Simplification time:         0.00 s
                                        |
|

    |
============================[ Search Statistics
    ]==============================
| Conflicts |          ORIGINAL          |          LEARNT
    | Progress |
|            |     Vars   Clauses  Literals  |     Limit   Clauses  Lit/Cl
    |           |
================================================================================

================================================================================

restarts               : 1
conflicts              : 0              (-nan /sec)
decisions              : 1              (0.00 % random) (inf /sec)
propagations           : 0              (-nan /sec)
conflict literals      : 0              (-nan % deleted)
Memory used            : 12.00 MB
CPU time               : 0 s

SATISFIABLE

SAT
-1 2 3 4 5 -6 -7 8 9 -10 0

WARNING: for repeatability, setting FPU to use double precision
==============================[ Problem Statistics
    ]===============================
|

    |
WARNING! DIMACS header mismatch: wrong number of clauses.
|  Number of variables:           10
                                              |
|  Number of clauses:             10
                                              |
|  Parse time:                  0.00 s
                                            |
|  Eliminated clauses:          0.00 Mb
                                           |
|  Simplification time:         0.00 s
                                            |
|

    |
==============================[ Search Statistics
    ]================================
| Conflicts |            ORIGINAL            |            LEARNT
    | Progress |
|            |     Vars   Clauses  Literals  |     Limit   Clauses  Lit/Cl
    |           |
================================================================================

================================================================================

restarts               : 1
conflicts              : 0              (0 /sec)
decisions              : 1              (0.00 % random) (674 /sec)
propagations           : 0              (0 /sec)
conflict literals      : 0              (-nan % deleted)
Memory used            : 12.00 MB
CPU time               : 0.001483 s

```

```
SATISFIABLE

SAT
−1 −2 −3 4 5 −6 7 8 9 −10 0
```

Problem 2.

Equivalence checking by implicit state enumeration.
Consider the product machine with shared input P=S1xS2 in your project 2.
Following the lecture on implicit state enumeration, use the implicit state enumeration to check the equivalence of S1 and S2.
The initial states of FFs in both circuits are all the equivalent states.
Use the Boolean algebra to do the reasoning.

Implicit State Enumeration

Reach-ability Analysis

Fig. 4: Reachability

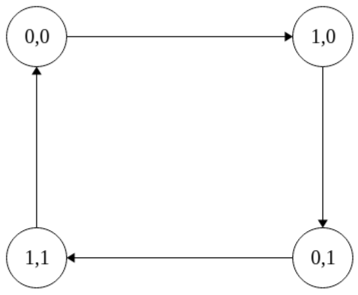
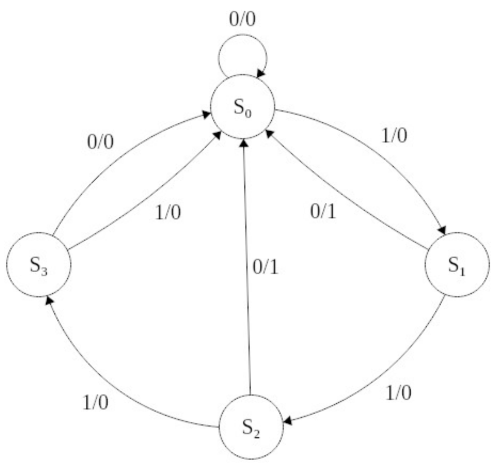
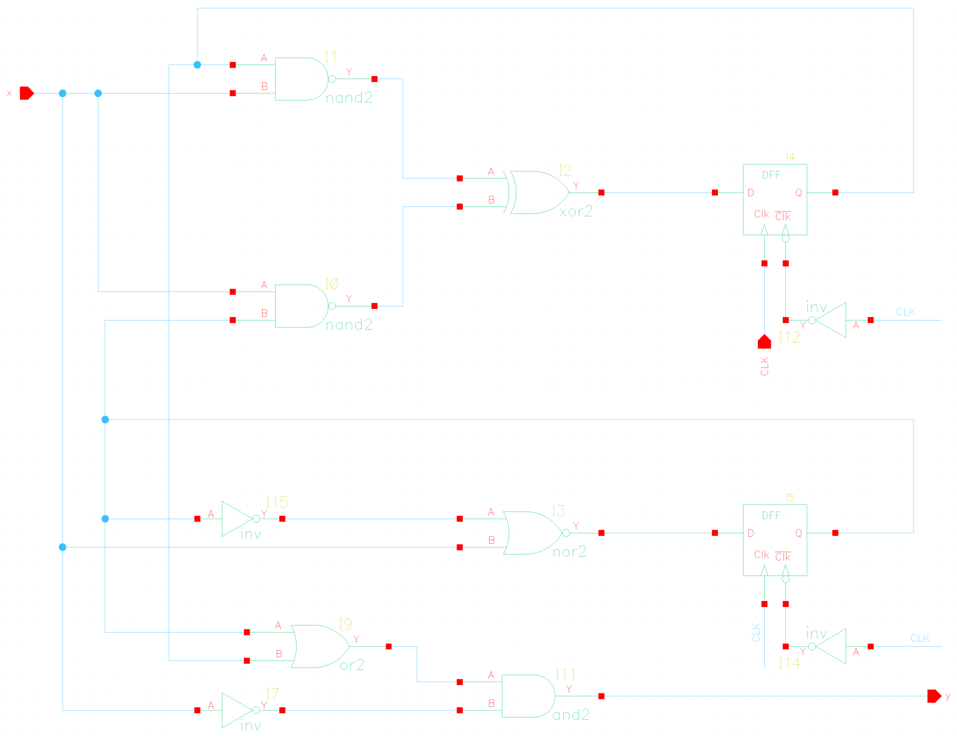


Fig. 5: State Diagram

Fig. 6: FSM from project 2





Equivalence Checking

$S0 = v1v2$

$R1 = av1v1' + av1v1' + av1v1' + av1v1'$

$R2 = av2v2' + av2v2' + av2v2' + av2v2$

$\exists a \exists v1 \exists v2 . S0 \wedge R1 \wedge R2$

$\exists a \exists v1 \exists v2 . (\neg v1 \neg v2) \wedge (av1v1' + av1v1' + av1v1' + av1v1') \wedge (av2v2' + av2v2' + av2v2' + av2v2')$

$= \exists a \exists v1 \exists v2 . (\neg av1v \neg 2v1' \neg v2' + a \neg v1 \neg v2 \neg v1'v2')$