# Lab 1 - Combinational Logic Circuits (I)

CDA 3201L-003
Jason Keene and Jacob Manfre
Submitted Jan 24th, 2015

## Purpose and Objectives

We set out to demonstrate the universality of NAND gates by implementing NOT, AND, and OR gates using only NAND gates. Additionally, we are going to simplify a boolean expression and implemented it with only NOT, AND, and OR gates.

## Component list

- Breadboard
- Wiring
- 5v power supply
- 3 x Quad 2-Input NAND Gate (74LS00)
- 1 x Hex Inverting Gate (74LS04)
- 1 x Quad 2-Input OR Gate (74LS32)
- 1 x Quad 2-Input AND Gate (74LS08)
- 4x LEDs
- 4x Resistors (470 Ohms 5%)

## Design

For the universality of NAND we did the boolean algebra to convert the three logic operators into using only NAND:

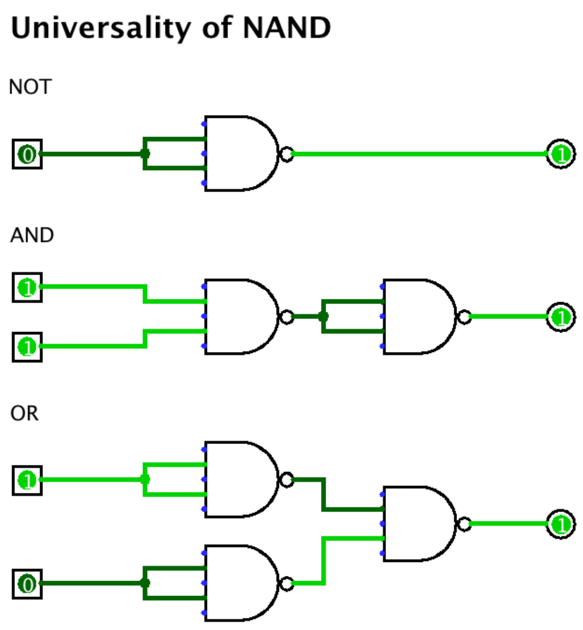
```
NOT : a'
      (aa)'    # idempotent law

AND : ab
      ab + ab           # idempotent law
      (ab + ab)''       # double negation law
      [(ab)'(ab)']'     # demorgan's law

OR : a + b
     (a + b)''          # double negation law
```

```
        (a'b')'          # demorgan's
        [(aa)'(bb)']'    # idempotent law
```

We then implemented these in logisim:



For the simplification we first implemented the full expression in python:

```python
def partb(x, y, z):
    return (
        not z and (
            not x and not y or
            x and not y
        ) or
        x and y or
        y and z and (y or z)
    )
```

We used this to generate the truth table:

| x | y | z | partb |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 |

| x | y | z | partb |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 |

We then attempted to simplify the expression with boolean algebra laws, checking our results against the truth table along the way.

```
F = z'(x'y' + xy') + xy + yz(y + z)
    z'(y'(x' + x)) + xy + yz(y + z)  # distribution law
    z'y' + xy + yz(y + z)            # inverse law
    z'y' + xy + yyz + yzz            # distribution law
    z'y' + xy + yz + yz              # idempotent law
    z'y' + xy + yz                   # idempotent law (sum_of_products)
```

Once we got to this point we decided to do a K-map to make sure the sum of products result was the same as what we came up with. There was two ways to do the K-map that were both equivalent. We implemented the result in python.

```python
def sum_of_products(x, y, z):
    return (
        not z and not y or      # 4 gates
        x and y or             # 2 gates
        y and z                # 1 gate
    )                          # total: 7 gates
```

At this point we noticed that it would take seven gates to implement the expression in hardware. It seemed obvious that we could reduce this further:

```
    (z + y)' + xy + yz               # demorgan's law
    (z + y)' + y(x + z)              # distribution law (simplification)
```

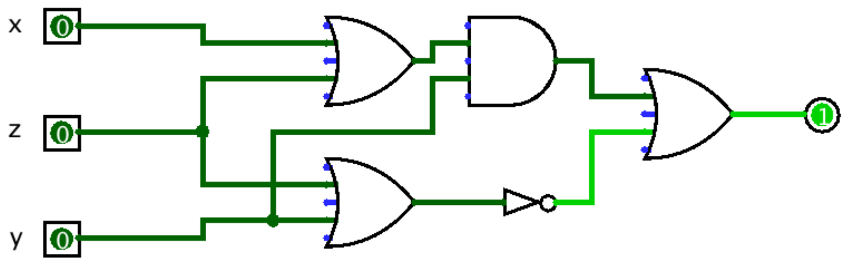
This gave us an expression we could implement with only five gates:

```python
def simplification(x, y, z):
    return (
        not (y or z) or        # 3 gates
        y and (x or z)         # 2 gates
    )                          # total: 5 gates
```

We then implemented this result in logisim:

### Simplification

(z + y)' + y(x + z)



## Test vectors and verification

Since there was a limited amount of inputs, we tested every possible input with the expected output in both logisim and with our hardware implementation.

## Discussion and conclusion

Through this exercise we have demonstrated that any combinational logic expression can be implemented with NAND gates. The only gates we haven't demonstrated to be implementable by NAND gates are:

- NOR gate
- XOR gate
- XNOR gate

NOR is simply a combination of NOT and OR. XOR is a combination of NOT AND, and OR (a'b + b'a). Finally, XNOR is a combination of NOT and XOR.

It is possible to simplify expressions into various logically equivalent forms to reduce the amount of components and increase reliability of your design.