ECE341 - Homework 1

Ton-Minh-Ky Tran

January 26, 2026

1 Introduction to the Logisim-evolution simulation software

2 Read the Logisim user guide

3 Basic logic gates implementation using 2-input NAND gate

We start off by defining $a \uparrow b$ to be the binary operator NAND of a and b. The specific implementation of each gate is illustrated in `24125102_3.circ`. The subsequent subsections will delineate the derivations of each implementation.

3.1 AND

One can easily see that $ab = \overline{a \uparrow b}$. As shown in **subsection 3.3**, one can implement by negation of x by taking the NAND of itself, i.e. $\overline{x} = x \uparrow x$. Thus the implementation of AND requires two NAND gates.

3.2 OR

Applying DeMorgan's law, we obtain $a + b = \overline{\overline{a}\overline{b}} = \overline{a} \uparrow \overline{b}$. The negations of a and b require one NAND gate each, taking the NAND of both negations thus totals to three NAND gates.

3.3 NOT

One can observe that the operator AND is idempotent, i.e. $a = aa$. Thus $\overline{a} = \overline{aa} = a \uparrow a$. The implementation of NOT thus requires one NAND gate.

3.4 XOR

A useful definition of $a \oplus b$ is $\overline{ab}(a + b)$. We obtain that $\overline{ab}(a + b) = \overline{ab}a + \overline{ab}b = \overline{\overline{\overline{ab}a} \cdot \overline{\overline{ab}b}}$ via DeMorgan's law. We require one NAND gate to evaluate $\overline{ab}$, two to evaluate its NAND with a and b, and one to evaluate the NAND of two products, totalling to a requirement of four NAND gates to implement XOR.

3.5 XNOR

We simply note that $a \odot b = \overline{a \oplus b}$. Using our implementation in **1.4**, we obtain a requirement of 5 NAND gates to implement XNOR.

4 Basic logic gates implementation using 2-input NOR gate

We start off by defining $a \downarrow b$ to be the binary operator NOR of a and b. The specific implementation of each gate is illustrated in `24125102_4.circ`. The subsequent subsections will delineate the derivations of each implementation.

4.1 AND

Applying DeMorgan's law, we obtain $ab = \overline{\overline{a} + \overline{b}} = \overline{a} \downarrow \overline{b}$. As shown in **subsection 4.3**, the negations of a and B require one NOR gate each, taking the NOR of both negations thus totals to three NOR gates.

4.2 OR

One can easily see that $a + b = \overline{a \downarrow b}$. Taking the negation of a NOR product requires two NOR gates.

4.3 NOT

One can observe that the operator OR is idempotent, i.e. $a = a + a$. Thus $\overline{a} = \overline{a + a} = a \downarrow a$. The implementation of NOT thus requires one NOR gate.

4.4 XOR

Once again the definition $a \oplus b = \overline{ab}(a+b)$ is useful. Thus $a \oplus b = \overline{ab}(a+b) = \overline{\overline{\overline{a} + \overline{b}} + \overline{a + b}} = (\overline{a} \downarrow \overline{b}) \downarrow (a \downarrow b)$. One can see that this implementation requires five NOR gates.

5 4-input XOR gate implementation using TTL 7400 IC

The TTL 7400 IC consists of four NAND gates, making it suitable as an implementation of a 2-input XOR gate. One can exploit the associativity of logical XOR, i.e. $A \oplus B \oplus C \oplus D = (A \oplus B) \oplus (C \oplus D)$, and thus a 4-input XOR gate can be built using three TTL 7400 ICs. An implementation is specified in `24125102_5.circ`.

6 Logic circuit of 2-input NAND gates to output 1 on certain input combinations and 0 otherwise

Based on the preconditions of the four input bits to output 1, a Karnaugh map is as follows:

$X3X2$ \ $X1X0$	00	01	11	10
00	1	0	1	0
01	0	1	1	0
11	0	1	1	0
10	1	0	1	0

Thus we obtain a disjunctive normal form $\overline{X2} \cdot \overline{X1} \cdot \overline{X0} + X1 \cdot X0 + X2 \cdot X0$. Applying the distributivity of logical OR over logical AND and DeMorgan's law, we obtain $\overline{X2 + X1} \cdot \overline{X0} + (X2 + X1)X0$. Using the definition of the XNOR gate, we obtain $(X2 + X1) \odot X0$. As shown in **subsection 3.2** and **subsection 3.5**, they require three and five NAND gates respectively, and so the implementation of the circuit requires eight NAND gates. An implementation is shown in `24125102_6.circ`.

7 8-bit lock and unlock button implementation using basic logic gates

The password we are to implement is the 8-bit form of 02, i.e. `00000010`. Since we have 8 input bits, we simply check 3 most significant bit pairs if they are both 0, and then check in the least significant bit pair if the first bit is set (by convention the zeroth bit is the least significant bit, so the first bit is the bit in the second position from right to left). To this end, we use the TTL 7436 IC (quad 2-input NOR gates), an AND gate with a negated input, and a 4-input AND gate to check if all eight bits are satisfied. Additionally, we use an AND gate to output `1` if the 8-bit password matches and the button is pressed. An implementation is shown in `24125102_7.circ`

8 Determine even parity bit for 7 bits of input data

One can observe that

$$\bigoplus_{i=1}^{n} a_i$$

evaluates to `0` if there are an even number j of a's, where $0 \leq j \leq n$, and evaluates to `1` otherwise. Since for even parity we wish to output a parity bit of `0` if there are even `1` bits, we thus simply take the XOR's of all bits and return the result. An implementation is shown in `24125102_8.circ`.