Assignment 1: 4-bit Full Adder

Introduction to Information Technology, Summer 2014

This assignment is worth 20 points ([0..10]:U, [10..15]:3, [15..18]:4, [18..20]:5) and contains two problems. Each student should provide an individual solution. Submissions can be made electronically on studentportalen.

Full Adder (11 points)

During the course, we have seen how to use XOR and AND logical gates in order to build a digital circuit which computes the sum of two bits. Figure 1 shows the truth table and the digital circuit that computes the outputs sum and cout from the input bits a and b.

In order to compute more complex arithmetic operations, we need to be able to integrate the half adder circuit (Figure 1b) into a bigger one. To that end, we should be able to take as input, besides a and b, a third input cin. This input represents the carried bit from previous computations. The output sum is the sum of three bits a, b and cin, while cout represents the carried out bit from that sum.

1. Draw the truth table that computes sum and cout from a, b and cin (2 points).

2. From the truth table, deduce the binary formula corresponding to sum and cout (3 points).

3. Draw the digital circuit corresponding to the full adder (3 points).

4. Using the full adder digital circuit, draw the digital circuit corresponding to a 4-bit full adder (3 points).

![Truth Table and Digital Circuit](image)

(a) Truth table of sum and cout

(b) Digital circuit: sum = a XOR b, cout = a AND b

Figure 1: Half adder
nand Definition
a \text{ nand } b = \text{ not } (a \text{ AND } b)

nor Definition
a \text{ nor } b = \text{ not } (a \text{ OR } b)

Conjunction Introduction
a = (a \text{ AND } a)

Disjunction Introduction
a = (a \text{ OR } a)

Double Negation
\text{ NOT } (\text{ NOT } a) = a

De Morgan (law)
The negation of a conjunction is the disjunction of the negations:

\text{ NOT } (a \text{ AND } b) = (\text{ NOT } (a) \text{ OR NOT } (b))

Reversibly, the negation of the disjunction is the conjunction of the negations:

\text{ NOT } (a \text{ OR } b) = (\text{ NOT } (a) \text{ AND NOT } (b))

Figure 2: Known results

Functional completeness (9 points)

A boolean formula can be derived from a truth table using three (basic) logical operators (or gates): NOT unary operator and AND and OR binary operators. For that reason, any binary logical operator can be defined using these three operators \{\text{not}, \text{and}, \text{or}\}\footnote{For instance, we can rewrite \text{xor} as follows: \text{XOR} = (a \text{ AND NOT } (b)) \text{ OR } (\text{NOT } (a) \text{ AND } b).}. This property is called functional completeness, and the set \{\text{not}, \text{and}, \text{or}\} is said functional complete.

1. We show in what follows that the singleton \{\text{nand}\} is a functional complete set. To that end, we use the rules listed in Figure 2. Using rules from the same set, prove yourself that the singleton \{\text{nor}\} is a functional complete set (3 points).

2. Rewrite the outputs \text{sum} and \text{c}_{\text{out}} formula in terms of the inputs \text{a}, \text{b} and \text{c}_{\text{in}} by only using \text{nor} operator (3 points).

3. Redraw the 1-bit full adder circuit by only using \text{nor} gates (3 points).

\{\text{nand}\} is functional complete

We show here that the singleton \{\text{nand}\} is a functional complete set. Since \{\text{not}, \text{and}, \text{or}\} is a functional complete set, any logical binary operator can be written as a formula where logical operators only from \{\text{not}, \text{and}, \text{or}\} are involved. In order to show that \{\text{nand}\} is a functional complete set, it suffices to show that any basic logic operator from the functional complete set \{\text{not}, \text{and}, \text{or}\} can be written in terms of \{\text{nand}\}.

Figure 2 lists a number of known boolean results that we use in this proof and that you can use yourself:

1. \text{ NOT } (a) = \text{ NOT } (a \text{ AND } a) = (a \text{ nand } a) \text{ from, successively, i) conjunction introduction and ii) definition of \text{nand}.}
2. \((a \lor b) = \neg(\neg(a \lor b)) = \neg(\neg a \land \neg b) = \neg(a \land \neg b) = (a \land \neg b) \land (\neg a \lor \neg b)\) from, successively, \(i\) double negation, \(ii\) De Morgan, \(iii\) \(\land\) definition and \(iv\) the result from 1 (how to rewrite \(\neg\) into \(\land\)).

3. \((a \land b) = \neg(\neg(a \land b)) = \neg(\neg a \lor \neg b) = (a \land \neg b) \land (\neg a \land b)\) from, successively, \(i\) double negation, \(ii\) \(\land\) definition and \(iii\) the result from 1 (how to rewrite \(\neg\) into \(\land\)).

Simulation

You can test your solutions either by building a hardware circuit composed of the different inputs, gates, wires and outputs or by using a digital circuit simulator. If you want to test your solution by simulation, you can use, among many free and proprietary tools, logisim, which you can download from the following address http://ozark.hendrix.edu/~burch/logisim/ Logisim is free of charge and can be installed on various platforms. An easy and quick video tutorial of this tool can be found on http://www.youtube.com/watch?v=ATPqpm1Vdw.