ICS Answer Sheet #8

Sakar Gopal Gurubacharya
s.gurubacharya@jacobs-university.de

Problem 8.1: combinational digital circuit

a. What is the Boolean expression implemented by the digital circuit?

We have 3 variables (input) and a single value (output) with many gates in between them (process).

$$\neg (\neg (\neg A \wedge \neg B) \wedge \neg (\neg (A \wedge B) \wedge C))$$

b. Derive algebraically step-by-step the disjunction (sum) of minterms from the Boolean expression implemented by the digital circuit.

$\neg (\neg (\neg A \wedge \neg B) \wedge \neg (\neg (A \wedge B) \wedge C))$	- 1a
$\neg (\neg \neg (A \vee B) \wedge \neg ((\neg A \vee \neg B) \wedge C))$	- de Morgan's Law (2x)
$\neg ((A \vee B) \wedge \neg ((\neg A \vee \neg B) \wedge C))$	- Double Negation Law
$\neg ((A \vee B) \wedge (\neg (\neg A \vee \neg B) \vee \neg C))$	- de Morgan's Law
$\neg ((A \vee B) \wedge (\neg \neg (A \wedge B) \vee \neg C))$	- de Morgan's Law
$\neg ((A \vee B) \wedge ((A \wedge B) \vee \neg C))$	- Double Negation Law
$\neg (A \vee B) \vee \neg ((A \wedge B) \vee \neg C)$	- de Morgan's Law
$(\neg A \wedge \neg B) \vee (\neg (A \wedge B) \wedge \neg \neg C)$	- de Morgan's Law
$(\neg A \wedge \neg B) \vee (\neg (A \wedge B) \wedge C)$	- Double Negation Law
$(\neg A \wedge \neg B) \vee ((\neg A \wedge C) \vee (\neg B \wedge C))$	- de Morgan's and Distributive Law
$(\neg A \wedge \neg B) \vee (\neg A \wedge C) \vee (\neg B \wedge C)$	- Removing unnecessary brackets.

A	B	C	(¬ A ∧ ¬ B)	(¬ A ∧ C)	(¬ B ∧ C)	(¬ A ∧ ¬ B) ∨ (¬ A ∧ C) ∨ (¬ B ∧ C)
0	0	0	1	0	0	1
0	0	1	1	1	1	1
0	1	0	0	0	0	0
0	1	1	0	1	0	1
1	0	0	0	0	0	0
1	0	1	0	0	1	1
1	1	0	0	0	0	0
1	1	1	0	0	0	0

Only taking the terms which results in 1, we get:

(¬ A ∧ ¬ B ∧ ¬ C) ∨ (¬ A ∧ ¬ B ∧ C) ∨ (¬ A ∧ B ∧ C) ∨ (A ∧ ¬ B ∧ C)

$m_0 + m_1 + m_3 + m_5$,

Therefore, sum of minterms: $\mathbf{m_0 + m_1 + m_3 + m_5}$**,**

Problem 8.2: full adder using different kinds of gates

$$S = (A\forall B) \forall C_{in}$$
$$C_{out} = (A \wedge B) \vee (C_{in} \wedge (A \forall B))$$

Truth Tables

S

A	B	C	$A\forall B$	$S = (A\forall B) \forall C_{in}$
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	0	1

C_{out}

A	B	C	$A \wedge B$	$A \forall B$	$(C_{in} \wedge (A \forall B))$	$C_{out} = (A \wedge B) \vee (C_{in} \wedge (A \forall B))$
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	1	0	0
0	1	1	0	1	1	1
1	0	0	0	1	0	0
1	0	1	0	1	1	1
1	1	0	1	0	0	1
1	1	1	1	0	0	1

a. **Write both functions as a disjunction of product terms.**

Disjunction of product terms means **sum of product terms**

Sum = OR
Product = AND

i.e., different **AND** expressions joined with **OR**
which is also called **DNF form.**

For DNF, we only take the interpretations that satisfies S and C_{out}, **i.e., result is 1.**

i. For S, that would be: $m_1 \lor m_2 \lor m_4 \lor m_7$.

$$\mathbf{S = (\neg A \land \neg B \land C) \lor (\neg A \land B \land \neg C) \lor (A \land \neg B \land \neg C) \lor (A \land B \land C)}$$

ii. For C_{out}, that would be: $m_3 \lor m_5 \lor m_6 \lor m_7$.

$$\mathbf{C_{out} = (\neg A \land B \land C) \lor (A \land \neg B \land C) \lor (A \land B \land \neg C) \lor (A \land B \land C)}$$

b. **Write both functions as a conjunction of sum terms.**

Conjunction of sum terms means **product of sum terms**

Sum = OR
Product = AND

i.e., different **OR** expressions joined with **AND**
which is also called **CNF form.**

For CNF, we only take the interpretations that dissatisfies S and C_{out}, **i.e., result is 0.**

i. For S, that would be: $m_0 \land m_3 \land m_5 \land m_6$.

$$\mathbf{S = (\neg A \lor \neg B \lor \neg C) \land (\neg A \lor B \lor C) \land (A \lor \neg B \lor C) \land (A \lor B \lor \neg C)}$$

iii. For C_{out}, that would be: $m_0 \land m_1 \land m_2 \land m_4$.

$$\mathbf{C_{out} = (\neg A \lor \neg B \lor \neg C) \land (\neg A \lor \neg B \lor C) \land (\neg A \lor B \lor \neg C) \land (A \lor \neg B \lor \neg C)}$$

c. **Write both functions using only not (¬) and not-and (↑) operations.**

i. **For S,**

$S = (A \veebar B) \veebar C_{in}$

For (A⊻ B),

$A \veebar B = (A \vee B) \wedge (\neg A \vee \neg B)$

$= (A \vee B) \wedge \neg (A \wedge B)$

We know that,

$A \uparrow B = \neg (A \wedge B)$

So,

$(A \veebar B) = ((A \uparrow (A \uparrow B)) \uparrow (B \uparrow (A \uparrow B)))$ *(From StackExchange)*

Then,

$S = (A \veebar B) \veebar C_{in}$

$= ((A \uparrow (A \uparrow B)) \uparrow (B \uparrow (A \uparrow B))) \veebar C_{in}$

$= ((((A \uparrow (A \uparrow B)) \uparrow (B \uparrow (A \uparrow B))) \uparrow (((A \uparrow (A \uparrow B)) \uparrow (B \uparrow (A \uparrow B))) \uparrow C_{in})) \uparrow (C_{in} \uparrow (((A \uparrow (A \uparrow B)) \uparrow (B \uparrow (A \uparrow B))) \uparrow C_{in})))$

Therefore,

$$\boxed{S = ((((A \uparrow (A \uparrow B)) \uparrow (B \uparrow (A \uparrow B))) \uparrow (((A \uparrow (A \uparrow B)) \uparrow (B \uparrow (A \uparrow B))) \uparrow C_{in})) \uparrow (C_{in} \uparrow (((A \uparrow (A \uparrow B)) \uparrow (B \uparrow (A \uparrow B))) \uparrow C_{in})))}$$

ii. **For C_{out},**

$C_{out} = (A \land B) \lor (C_{in} \land (A \veebar B))$

- **For $(A \veebar B)$,**

  $A \veebar B = (A \lor B) \land (\neg A \lor \neg B)$

  $= (A \lor B) \land \neg (A \land B)$

  We know that,

  $A \uparrow B = \neg (A \land B)$

  So,

  $(A \veebar B) = ((A \uparrow (A \uparrow B)) \uparrow (B \uparrow (A \uparrow B)))$ *(From StackExchange)*

- **For $(A \land B)$**

  $(A \land B) = ((A \uparrow B) \uparrow (A \uparrow B))$ *(From Wikipedia)*

- **For $(A \lor B)$**

  $(A \lor B) = ((A \uparrow A) \uparrow (B \uparrow B))$ *(From Wikipedia)*

Now, all operators in C_{out} expression are covered for with NAND gates, we can now express the entire expression with just NAND gates.

$C_{out} = (A \wedge B) \vee (C_{in} \wedge (A \veebar B))$

$= (((A \uparrow B) \uparrow (A \uparrow B)) \vee (C_{in} \wedge ((A \uparrow (A \uparrow B)) \uparrow (B \uparrow (A \uparrow B)))))$

For now, dealing with the expression on the **right side** of the **OR** operator.

So, we have:

$= (C_{in} \wedge ((A \uparrow (A \uparrow B)) \uparrow (B \uparrow (A \uparrow B))))$

$= ((C_{in} \uparrow ((A \uparrow (A \uparrow B)) \uparrow (B \uparrow (A \uparrow B)))) \uparrow (C_{in} \uparrow ((A \uparrow (A \uparrow B)) \uparrow$
$(B \uparrow (A \uparrow B)))))$

Now, finally combining all and changing the **OR** operator to **NAND.**

So, we have:

$= (((A \uparrow B) \uparrow (A \uparrow B)) \vee ((C_{in} \uparrow ((A \uparrow (A \uparrow B)) \uparrow (B \uparrow (A \uparrow B)))) \uparrow$
$(C_{in} \uparrow ((A \uparrow (A \uparrow B)) \uparrow (B \uparrow (A \uparrow B))))))$

$= ((((A \uparrow B) \uparrow (A \uparrow B)) \uparrow ((A \uparrow B) \uparrow (A \uparrow B))) \uparrow (((C_{in} \uparrow ((A \uparrow (A \uparrow B)) \uparrow$
$(B \uparrow (A \uparrow B)))) \uparrow (C_{in} \uparrow ((A \uparrow (A \uparrow B)) \uparrow (B \uparrow (A \uparrow B))))) \uparrow$
$((C_{in} \uparrow ((A \uparrow (A \uparrow B)) \uparrow (B \uparrow (A \uparrow B)))) \uparrow (C_{in} \uparrow ((A \uparrow (A \uparrow B)) \uparrow$
$(B \uparrow (A \uparrow B)))))))$

Therefore,

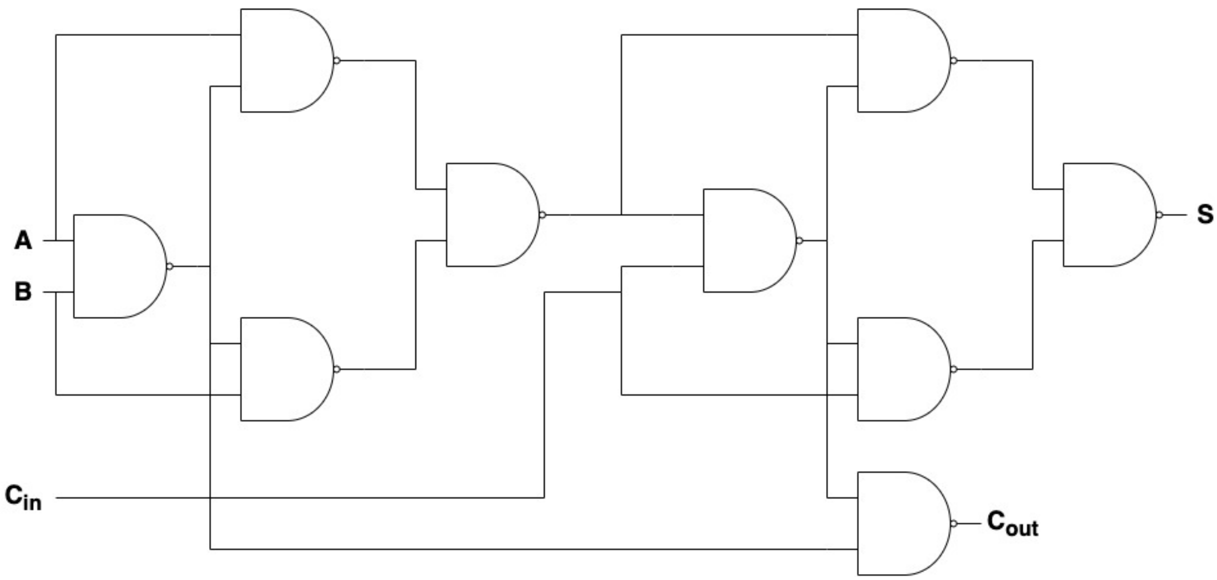
$C_{out} = ((((A \uparrow B) \uparrow (A \uparrow B)) \uparrow ((A \uparrow B) \uparrow (A \uparrow B))) \uparrow (((C_{in} \uparrow ((A \uparrow (A \uparrow B)) \uparrow$

$(B \uparrow (A \uparrow B)))) \uparrow (C_{in} \uparrow ((A \uparrow (A \uparrow B)) \uparrow (B \uparrow (A \uparrow B))))) \uparrow$

$((C_{in} \uparrow ((A \uparrow (A \uparrow B)) \uparrow (B \uparrow (A \uparrow B)))) \uparrow (C_{in} \uparrow ((A \uparrow (A \uparrow B)) \uparrow$

$(B \uparrow (A \uparrow B)))))))$

d. **In a digital circuit, we can easily reuse common terms. Draw a small digital circuit implementing S and C_{out} using NAND gates only.**



(Gates Circuit made from ***draw.io****)*

Problem 8.3: haskell fizzbuzz

a. **Write a Haskell function fizzbuzz :: Integer -> String that takes a positive integer and returns the number rendered as a string or one of the strings "fizz", "buzz", or "fizzbuzz", following the rules defined above.**

b. **Using foldr, write a simple expression that returns the fizzbuzz sequence as a list of strings for the numbers in the range 1 to 16. Do not use list comprehensions or other higher order functions or lambda functions.**

c. **Using foldl, write a simple expression that returns the fizzbuzz sequence as a list of strings for the numbers in the range 1 to 16. Do not use list comprehensions or other higher order functions (and ideally no lambda functions but you may use flip).**

(fizzbuzz.txt present in the same .zip file)

fizzbuzz.hs — Assignment 8

fizzbuzz.hs

```haskell
{-
Having fizzbuzz as the first condition is necessary,
otherwise conditions before it would be satisfied before and end that "iteration"
-}

fizzbuzz :: Int -> String
fizzbuzz n =

   if mod n 3 == 0 && mod n 5 == 0 then
   "fizzbuzz"

   else if mod n 3 == 0 then
   "fizz"

   else if mod n 5 == 0 then
   "buzz"

   else
   show n

--using foldr.
map' f = foldr ((:) . f) [] [1..16]
--type "map' fizzbuzz" in ghci

--using foldl.
map'' f = foldl (flip ((:) . f)) [] [16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1]
--type "map'' fizzbuzz" in ghci
```

References

1. *XOR in NAND taken from: https://math.stackexchange.com/questions/38473/is-xor-a-combination-of-and-and-not-operators*

2. *Every gate in NAND gates taken from: https://en.wikipedia.org/wiki/NAND_logic*

3. *Digital circuit made from: https://app.diagrams.net/*