COMP2611: Computer Organization

Introduction to Combinational Logic

Overview

- You will learn about the combinational logic circuits
  - Do not have internal states (i.e. memoryless),
  - The output is solely determined by the present input and the circuit,
  - Can be specified with a truth table or a logic equation (in Boolean algebra expression).
- Vs Sequential Logic circuits (the next tutorial)
  - Have memory,
  - The output depends on both the current input and the value stored in the memory of the circuits (called state).

Combinational logic

Boolean algebra

- review of the Boolean algebra
- the Sum of Product (SoP) representation
- from Boolean algebra to circuit
- PLA implementation
- K-Maps

Exercises

Why Boolean Algebra?

- The Input-Output relationship of any combinational logic circuit can be completely specified using either
  - a truth table,
  - or a Boolean algebra expression.
- When there are N inputs, the truth table would require as many as 2^N entries.
- The Boolean algebra expression does not have this “cardinality explosion” problem.

Review of Boolean Algebra 1

- Boolean algebra consists of
  - Boolean variables (with values equal to either '0' or '1'),
  - and binary operators AND ('•'), OR ('+'), NOT ('‾') or (' ' ').

- The AND, OR, and NOT operations form a functionally complete set, as they can specify any logic function.

Review of Boolean Algebra 2

- **Identity laws**:

$$A+0=A \qquad A\cdot 1=A$$

- **Annihilator (or Zero and one) laws**:

$$A+1=1 \qquad A\cdot 0=0$$

- **Complement laws**:

$$A+\overline{A}=1 \qquad A\cdot \overline{A}=0$$

- **Commutativity laws**:

$$A+B=B+A \qquad A\cdot B=B\cdot A$$

- **Associativity laws**:

$$A+(B+C)=(A+B)+C \qquad A\cdot(B\cdot C)=(A\cdot B)\cdot C$$

- **Distributivity laws**:

$$A\cdot(B+C)=(A\cdot B)+(A\cdot C) \qquad A+(B\cdot C)=(A+B)\cdot(A+C)$$

- **Idempotence**:

$$A + A = A \qquad\qquad A \cdot A = A$$

- **Absorption laws**:

$$A + (A \cdot B) = A \qquad\qquad A \cdot (A + B) = A$$

- **De Morgan Laws**:

$$\overline{A + B} = \overline{A} \cdot \overline{B}$$

$$\overline{A \cdot B} = \overline{A} + \overline{B}$$

Truth table to Boolean representation 1

- Any logic function can be expressed as a two-level representation, either as
  - the Sum-of-Products (SoP) representation,
  - or as the Product-of-Sums (PoS) representation.

- Example: Assume the truth table for a circuit is given as:

Inputs		Outputs	
In0	**In1**	**Out0**	**Out1**
0	0	1	0
0	1	1	1
1	0	0	1
1	1	0	0

Express the truth table using SoP representations in Boolean Algebra.

Truth table to Boolean representation 2

- For each output (Out0, Out1),

  I. Observe the rows that the output has a value of '1'.

  - For Out0, the 1's are at the 1st and the 2nd rows.

In0	In1	Out0
0	0	1
0	1	1

  II. Write the minterms for the inputs such that the minterms will give 1's for the input patterns in the same rows.

  - For Out0, the two 1's correspond to the input patterns (in0=0,in1=0) and (in0=0,in1=1). The two minterms that will give 1's are $\overline{\text{in0}} \cdot \overline{\text{in1}}$ and $\overline{\text{in0}} \cdot \text{in1}$

  III. Write the outputs as the OR operation of the minterms found in step two. For Out0, we have $\text{out0} = \overline{\text{in0}} \cdot \overline{\text{in1}} + \overline{\text{in0}} \cdot \text{in1}$

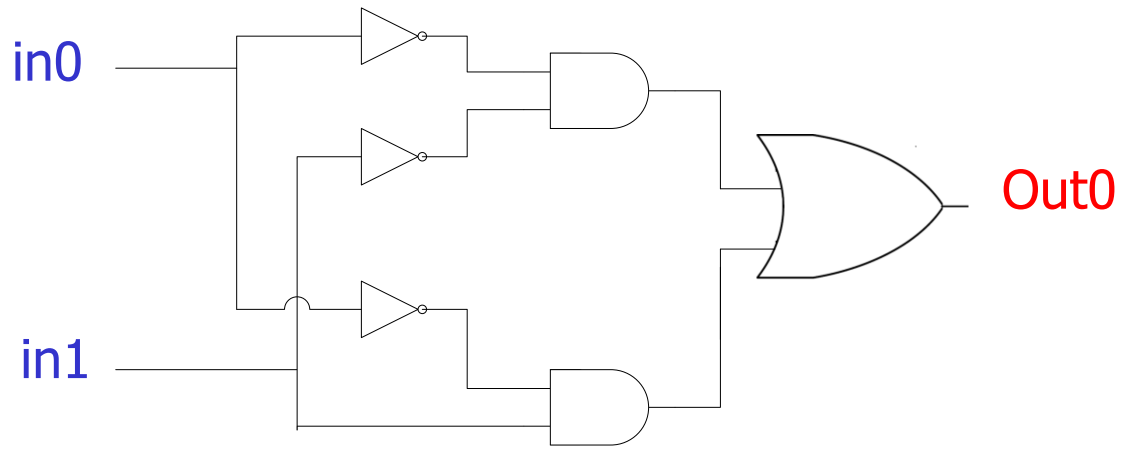
Truth table to Boolean representation 3

- By following the above steps. The required expressions for the two outputs (Out0, Out1) are $\mathrm{out0} = \overline{\mathrm{in0}} \cdot \overline{\mathrm{in1}} + \overline{\mathrm{in0}} \cdot \mathrm{in1}$ and $\mathrm{out1} = \overline{\mathrm{in0}} \cdot \mathrm{in1} + \mathrm{in0} \cdot \overline{\mathrm{in1}}$ respectively.

- Could the expressions for Out0 and Out1 be further simplified using the laws on slide 6?

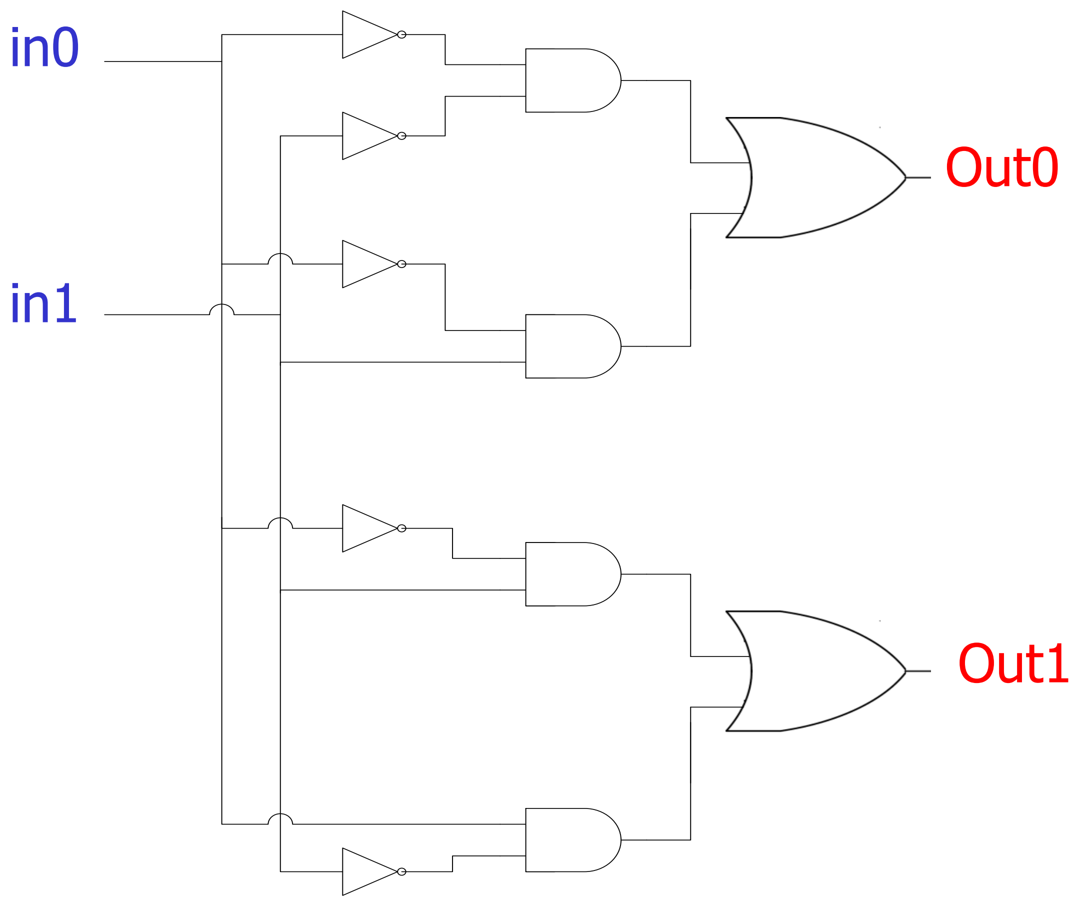
Reducing a Boolean expression to a circuit 1

- The expression $out0 = \overline{in0} \cdot \overline{in1} + \overline{in0} \cdot in1$ can be viewed as performing the OR operation on two ANDed minterms $\overline{in0} \cdot \overline{in1}$ and $\overline{in0} \cdot in1$.
- The circuit is as follows. Mind the two AND gates that correspond to the AND expressions and the single OR gate that corresponds to the OR expression.



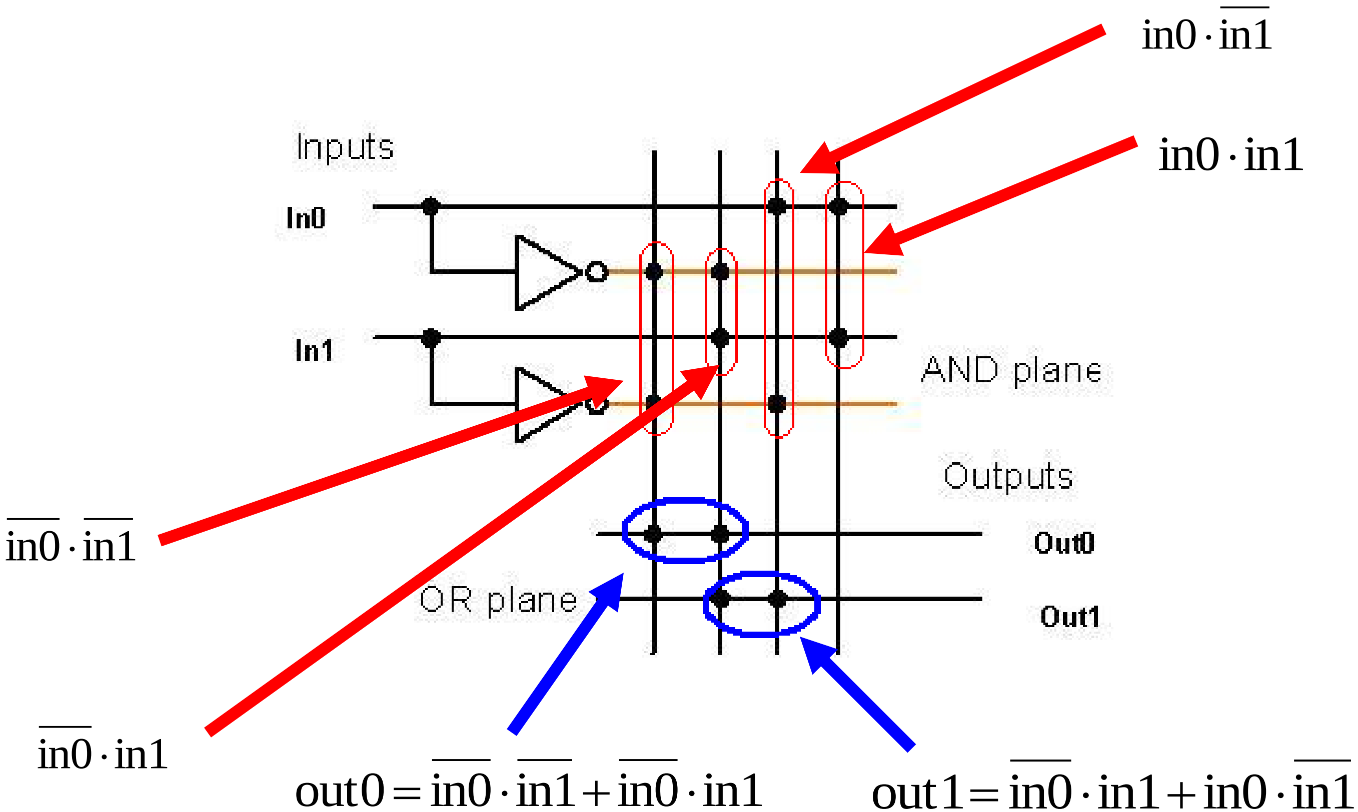
- What will be the impact on the circuit if we use the simplified expression (mentioned in the last slide) to build the circuit.

- Using the same approach, the expression $out1 = \overline{in0} \cdot in1 + in0 \cdot \overline{in1}$ can also be drawn. Combine it with the previous figure we have the overall circuit:



PLA implementation of a Boolean expression

- The same circuit can be equivalently represented by a programmable logic array (PLA) circuit.



K-Maps

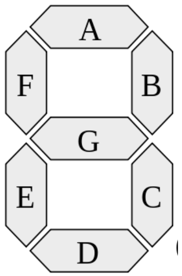
- K-Map is a graphical representation of the truth table or logic function
- In a K-map each cell represents one possible minterm
- Cells are arranged following a Gray code i.e., two adjacent cells are such that the corresponding minterms differ in only one variable
- Simplify expression by finding largest size groups of adjacent cells at 1 in the K-Map
  - Can only group 2^n adjacent cells where n = 0, 1, 2, 3, 4, …
  - Table is a toroid (i.e., rightmost cells are adjacent to the leftmost cells and topmost cells are adjacent to bottom cells)
- Example: Simplify F = AB′ + AB + A′B

A \ B	**0**	**1**
0	A′B′	A′B
1	AB′	AB

A \ B	**0**	**1**
0	0	1
1	1	1

F = B + A

- Example: Consider a 7-segment digital display which displays a hexadecimal digit. Each segment is represented by a logic function



- That is, 4 inputs i_3, i_2, i_1, i_0 to represent values 0, …, 9, a, b, c, d, e, f (to avoid confusion between 0, and 8 on one hand and D and B respectively, on the other hand, we use miniscule b and d representation)
- What is the truth table for segment C?
- Also, use a K-Map to simplify the equation

K-Maps

- Truth Table for segment C:

	Inputs				Output
Hexadecimal Digit	i_3	i_2	i_1	i_0	C
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
A	1	0	1	0	1
b	1	0	1	1	1
C	1	1	0	0	0
d	1	1	0	1	1
E	1	1	1	0	0
F	1	1	1	1	0

K-Maps

- K-Map:

i_3i_2 \ i_1i_0	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	0	1	0	0
10	1	1	1	1

- $C = i_3' \, i_1' + i_3' \, i_0 + i_3' \, i_2 + i_1' \, i_0 + i_3 \, i_2'$

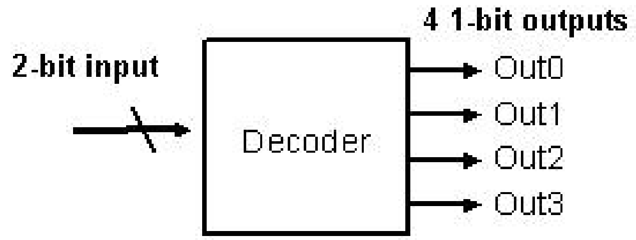
Combinational logic

Boolean algebra

- review of the Boolean algebra
- the Sum of Product (SoP) representation
- from Boolean algebra to circuit
- PLA implementation
- K-maps

Exercises

Question 1: A decoder takes a single N-bit input and outputs 2^N 1-bit signals. The 1-bit output corresponds to the N-bit input bit pattern is true while all other outputs are false. The following figure shows a block diagram for a 2-to-4 decoder.

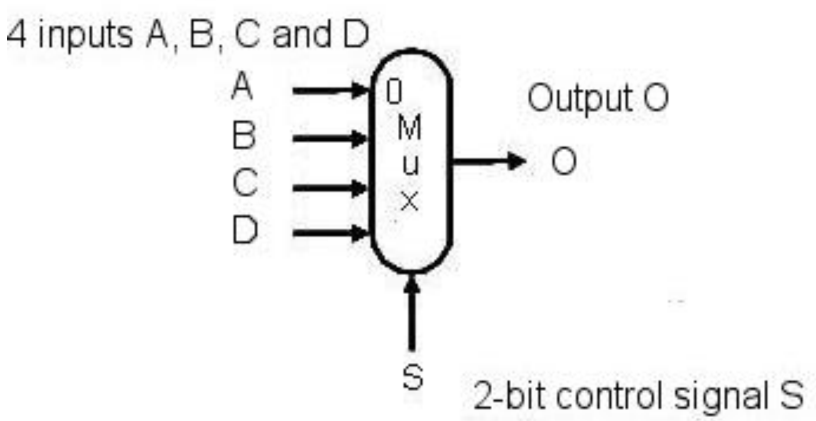


A 2-bit decoder

Exercises

- Why a 2-bit input can generate 4 outputs in the decoder?
- If the input bits are 11, what will happen to the outputs of the decoder?
- Is it possible to have more than one outputs asserted?
- Name two potential uses of the decoder.
- Implement the decoder using Logisim

Question 2: A multiplexor is a devices that given the control signal, selects one of the inputs to be forwarded to the output. The following figure shows a 4-input multiplexor.



Exercises

- If the inputs A/B are 32-bit in width, what is the data width of the Output O?
- What is the maximum number of inputs if the control signal is 10-bit in width?
- What is the bit-width of the control signal for the multiplexor if there are 9 inputs?

Conclusion

- Today we have reviewed:
  - simple Boolean algebra and the related laws,
  - reducing truth table to the canonical Sum-of-Products form,
  - converting simple Boolean algebra expressions to circuits,
  - simple combinational logic circuits,
  - K-maps.