Review with Examples

CS207: Digital Logic

Jialin Liu

Department of Computer Science and Engineering (CSE)
Southern University of Science and Technology (SUSTech)

23 December 2022

Course Plan

- Week 1 (Sept. 9): Lecture 1 - Binary Numbers
- Week 2 (Sept. 16): Lecture 2 - Boolean Algebra and Logic Gates
- Week 3 (Sept. 23): Lecture 3 - Gate-Level Minimisation
- Week 4 (Sept. 30): Lecture 4 - Two/Multi-Level Implementation
- ——— (Oct. 9): Assignment 1 Analysis & Lecture 5 - Combinational Logic
- Week 5 (Oct. 14): Lecture 6 - Standard Components
- Week 6 (Oct. 21): Lecture 7 - Arithmetic Circuits
- Week 7 (Oct. 28): Lecture 8 - Memory Elements: Latches and Flip-flops
- Week 8 (Nov. 4): Lecture 8 (cont'd) & Lecture 9 - Sequential Logic
- **Week 9 (Nov. 11): No course (University Sports Day)**

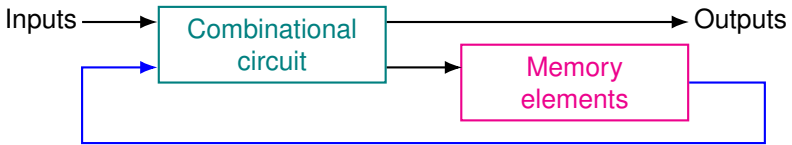
Course Plan

- **Week 10 (Nov. 18): No course according to University's calendar**
  QA Session & Assignment 2 Analysis
- Week 11 (Nov. 25): Open-book in-class test
- Week 12 (Dec. 2): Mid-term test Analysis & Assignment 3 Analysis
- Week 13 (Dec. 9): Lecture 10 - Registers
- Week 14 (Dec. 16): Lecture 11 - Counters
  Project Inspection #1 in lab
- Week 15 (Dec. 23): Review with Exercise and Solutions
- **Submission deadline of Assignment 4: 23:55, Dec. 27 (Beijing time)**
- Week 16 (Dec. 30): Assignment 4 Analysis
  Project Inspection #2 in lab
- **Final exam: 2-4pm, Jan. 4 (Beijing time)**
- **Online assessment: 2:30-3:30pm, Jan. 4 (Beijing time)**

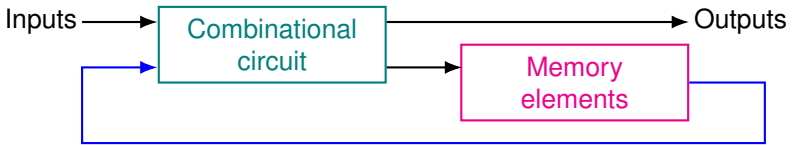
- Most digital systems are build with sequential logic.

- Most digital systems are build with sequential logic.
  - How to analyse / design **sequential circuits**? **(Lecture 9)**
  - Commonly used sequential circuits: register **(Lecture 10)** and counter **(Lecture 11)**

- Most digital systems are build with sequential logic.
  - How to analyse / design **sequential circuits**? **(Lecture 9)**
  - Commonly used sequential circuits: register **(Lecture 10)** and counter **(Lecture 11)**
- A sequential circuit is composed of feedback path,
  - **memory elements (Lecture 8)**
    → Different types of memory elements, their characteristic equations/tables.
  - **combinational circuits**



- Most digital systems are build with sequential logic.
  - How to analyse / design **sequential circuits**? **(Lecture 9)**
  - Commonly used sequential circuits: register **(Lecture 10)** and counter **(Lecture 11)**
- A sequential circuit is composed of feedback path,
  - **memory elements** **(Lecture 8)**
    → Different types of memory elements, their characteristic equations/tables.
  - **combinational circuits**
    - How to analyse / design combinational circuits? **(Lecture 5)**
      → Basic notions & logic gates **(Lectures 1 – 2)**; Gate-level minimisation **(Lecture 3)**; Gate-level implementation **(Lecture 4)**
    - Standard components: Decoder, encoder, multiplexer, ... **(Lecture 6)**
    - Arithmetic circuits: adder, subtractor, multiplicator, ... **(Lecture 7)**



Lectures 8-11
时序逻辑电路
分析与设计
&
记忆模块元件

Lectures 5-7
组合逻辑电路
分析与设计
&
基本模块元件

Lectures 3 & 4
门级电路
化简与实现

Lectures 1 & 2
基础概念和公理

Sequential circuits

- Key difference compared to combinational circuits: memory elements, feedback path
- Representation of sequential circuits: logic diagram, function table
- Memory elements: latch & flip-flop (key difference: level sensitive devices vs. edge-sensitive devices)
  - (Controlled) SR latch, (Controlled) D latch, DFF (DFF, diagram, function table), JKFF
- State reduction & state assignment
- Analysis and design of sequential circuits (specification -> a logic diagram)

Combinational circuits

- Analysis of combinational circuits: given a diagram, output Boolean functions or truth table
- Design of combinational circuits: specification -> a logic diagram / Boolean functions
- Standard components: encoder (Priority encoder), decoder (minterm generator), multiplexer (Boolean function generator), magnitude comparator, binary adder (carry, half adder, full adder, carry lookahead logic), binary subtractor (borrow, half subtractor, full subtractor, overflow), binary multiplicator (implementation)

Gate-level design

- Gate-level minimisation: Boolean Algebra; Karnaugh map (2/3/4-vars K-maps, don't care)
- Implement circuits: Two/multi-level implementation using NAND, NOR and XOR gates

- Number Systems & Conversions
- Complement of numbers
- Representation of Data -- Code (8421BCD, Gray Code)

- Boolean Algebra & Boolean functions
- SOP/POS forms, sum of minterms/maxterms
- Logic operators and gates

Gate-level implementation (Lecture 4)

- **Two/multi-level implementation** using `NAND`, `NOR` and `XOR` (Lecture 4)
  → based on gate-level minimisation and logic gates

- **Gate-level minimisation**
  - **Karnaugh map** (Lecture 3)
    - 2/3/4-vars K-maps
    - don't care
  - **Boolean Algebra** (Lecture 2)
    - Besides simplifying a Boolean function, expressing a function in sum of minterms / product of maxterms also requires Boolean Algebra.

  → based on basic notions in Lectures 1&2 (number systems, conversions; logic operators and gates)

Recall Boolean Algebra (Lecture 2)

Exercise

Express the function $F(A,B,C,D) = B'D + A'D + BD$ as a sum of minterms with $\sum$ and as a product of maxterms with $\prod$.

Recall Boolean Algebra (Lecture 2)

Exercise

Express the function $F(A,B,C,D) = B'D + A'D + BD$ as a sum of minterms with $\sum$ and as a product of maxterms with $\prod$.

Solution:

$$
\begin{aligned}
F(A,B,C,D) &= B'D + A'D + BD \\
&= (A+A')B'(C+C')D + A'(B+B')(C+C')D + (A+A')B(C+C')D \\
&= AB'CD + A'B'CD + AB'C'D + A'B'C'D + \\
&\quad A'BCD + A'B'CD + A'BC'D + A'B'C'D + \\
&\quad ABCD + A'BCD + ABC'D + A'BC'D \\
&= \sum(1,3,5,7,9,11,13,15) \\
&= \prod(0,2,4,6,8,10,12,14)
\end{aligned}
$$

Recall K-map (Lecture 3)

Exercise

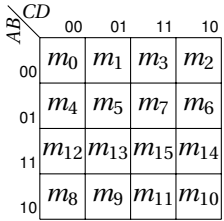
Simplify the following Boolean function F together with the don't-care conditions d:
$F(A,B,C,D) = \sum(2,4,10,12,14,15)$, $d(A,B,C,D) = \sum(0,1,3)$ using **K-map**. Truth table is not necessary.

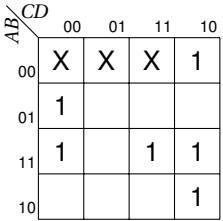
Recall K-map (Lecture 3)

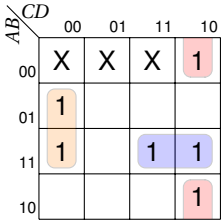
Exercise

Simplify the following Boolean function F together with the don't-care conditions d:
$F(A,B,C,D) = \sum(2,4,10,12,14,15)$, $d(A,B,C,D) = \sum(0,1,3)$ using **K-map**. Truth table is not necessary.

Solution:







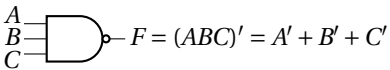
$$F(A,B,C,D) = BC'D' + ABC + B'CD'$$

Two/multi-level implementation: NAND and NOR Circuits (Lecture 4)

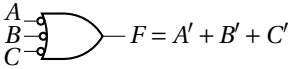
- NAND gates (与非门) and NOR gates (或非门) are called **universal gate**s or **universal building block**s. Any type of gates or logic functions can be implemented by these gates.
- Two- and multi-level implementations:
  - Procedure of two-level implementations using NAND or NOR gates.
  - Procedure of multi-level implementations using NAND or NOR gates.
- Exclusive-OR gate for constructing error detection circuits.

- A convenient way to implement a Boolean function with NAND gates:
  1. Obtain its simplified Algebraic expression.
  2. Convert the Algebraic expression to NAND logic.
- To facilitate the conversion to NAND logic, it is convenient to define an alternative graphic symbol for the gate.

AND-invert:



Invert-OR:



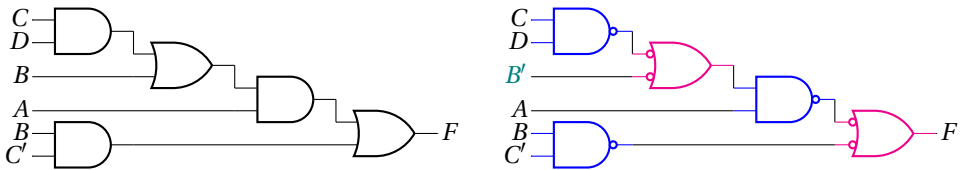
Two/multi-level implementation (Lecture 4)

NAND Circuits: Procedure

- A Boolean function can be implemented with **two levels of NAND gates**.
  1. Simplify the Algebraic expression and express it in sum-of-products form.
  2. Draw a NAND gate for each product term of the expression that has at least two literals. The inputs to each NAND gate are the literals of the term. This procedure produces a group of **first-level gates**.
  3. A term with a single literal requires an inverter in the first level. However, if the single literal is complemented, it can be connected directly to an input of the second-level NAND gate.
  4. Draw a single gate using the AND-invert or the invert-OR graphic symbol **in the second level**, with inputs coming from outputs of first-level gates.

C.f. Example of Lecture 4 (pages 25-33)

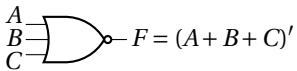
- Convert the following multilevel AND–OR diagram into an all-NAND diagram using mixed notation.
  1. Convert all AND gates to NAND gates with AND-invert graphic symbols.
  2. Convert all OR gates to NAND gates with invert-OR graphic symbols.
  3. Check all the bubbles in the diagram. For every bubble that is not compensated by another small circle along the same line, insert an inverter (a one-input NAND gate) or complement the input literal.



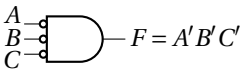
NOR Circuits (Lecture 4)

- To facilitate the conversion to NOR logic, it is convenient to define an alternative graphic symbol for the gate.

OR-invert:



Invert-AND:



- **Procedure:**
  - A two-level implementation with NOR gates requires that the function be simplified into product-of-sums form.
  - Change the OR gates to NOR gates with OR-invert graphic symbols and the AND gate to a NOR gate with an invert-AND graphic symbol.
- **Remark:** The equivalent AND-OR diagram can be recognised from the NOR diagram by removing all the bubbles.

Combinational Circuits (Lectures 5 – 7)

- Combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs.
  - **Procedure of analysing combinational circuits** (Lecture 5)
    - Given a diagram, output Boolean functions or truth table
  - **Procedure of designing combinational circuits** (Lecture 5)
    - Specification -> a logic diagram / Boolean functions
  - **Standard components** (Lecture 6)
    - Encoder (Priority encoder), decoder (minterm generator), multiplexer (Boolean function generator) → Very important!
    - magnitude comparator
  - **Arithmetic circuits** (Lecture 7)
    - Binary adder (carry, half adder, full adder, carry lookahead logic), binary subtractor (borrow, half subtractor, full subtractor, overflow), binary multiplicator (implementation)

Combinational Circuits (Lecture 5)

Procedure of analysing combinational circuits

- **Input**: a logic diagram.
  1. **Make sure that the given circuit is combinational and not sequential.**
     - The diagram of a combinational circuit has logic gates with no feedback paths or memory elements.
     - **Feedback path**: a connection from the output of one gate to the input of a second gate whose output forms part of the input to the first gate.
  2. **Obtain the output Boolean functions from the given diagram:**
     - 2.1 Label all gate outputs that are a function of input variables with arbitrary symbols – but with meaningful names (i.e., only inputs, no other intermediate variables). Determine the Boolean functions for each gate output.
     - 2.2 Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Determine their Boolean functions.
     - 2.3 Repeat the process outlined in step 2.2 until the outputs of the circuit are obtained.
     - 2.4 By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.
- **Output**: a set of Boolean functions.

←C.f. Example of Lecture 5 (pages 14-19)

- Design of combinational circuits: develop a logic circuit diagram or a set of Boolean functions from specification of the design objective.
  - Input: a specification of the design objective.
  - Output: a logic circuit diagram or a set of Boolean functions.
- Steps:
  - From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.
  - Derive the truth table that defines the required relationship between inputs and outputs.
  - Obtain the simplified Boolean functions for each output as a function of the input variables.
  - Draw the logic diagram and verify the correctness of the design (manually or by simulation).

←C.f. Example of Lecture 5 (pages 21-26)

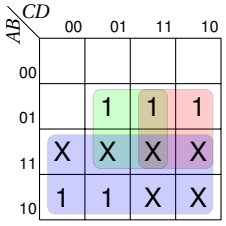
Exercise

Convert from BCD code to excess-3 code (i.e., self-complementary BCD code used to represent the decimal numbers).

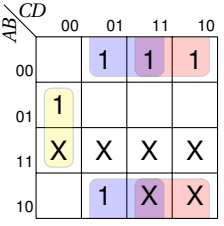
Decimal	BCD code				Excess-3 code			
digit	A	B	C	D	w	x	y	z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Procedure:

1. From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.
   → Each code uses 4 bits, therefore 4 input bits and 4 output bits.
2. Derive the truth table that defines the required relationship between inputs and outputs.
   → We already have the truth table :)
3. Obtain the simplified Boolean functions for each output as a function of the input variables.
   → Use K-map!
4. Draw the logic diagram and verify the correctness of the design.

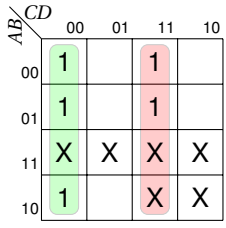


$$w = A + BD + BC$$

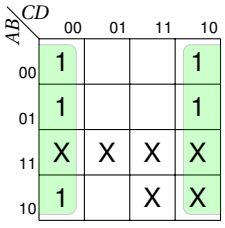


$$x = BC'D' + B'D + B'C$$

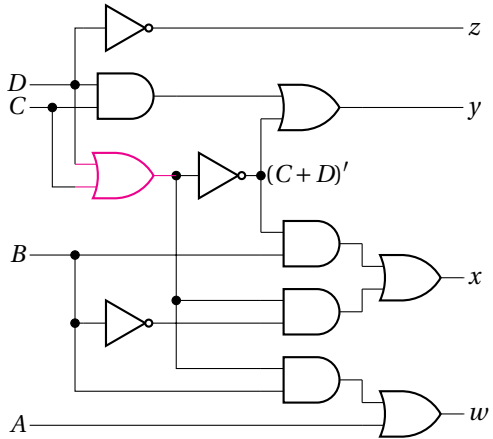
$$y = C'D' + CD$$


AB
CD
00
01
11
10
00
01
11
10
1
1
X
X
1
1
X
X
X
X
X
X

$$z = D'$$


AB
CD
00
01
11
10
00
01
11
10
1
1
X
1
1
X
X
X
X
X
X

- We manipulate these Boolean functions to reuse common gates:
  - $w = A + BC + BD = A + B(C + D)$.
  - $x = B'C + B'D + BC'D' = B'(C + D) + B(C + D)'$.
  - $y = CD + C'D' = CD + (C + D)'$.
  - $z = D'$.



$$w = A + B(C + D), \quad x = B'(C + D) + B(C + D)', \quad y = CD + (C + D)', \quad z = D'.$$

Standard Components (Lecture 6)

- Standard components: combinational circuits that are employed extensively in the design of digital systems.
  - Decoders
  - Encoders
  - Multiplexers
  - Comparators.
- Available in integrated circuits as medium-scale integration (MSI) circuits, also used as standard cells in complex very large-scale integrated (VLSI) circuits.

Standard Components (Lecture 6)

Decoder (译码器)

- A **decoder** is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines.
  - The selected output is identified either by a `1`, when all other outputs are `0`, or by a `0` when all other outputs are `1`.
  - The basic function of a decoder having n inputs is to select 1-out-of-2^n output lines.

- Main applications:
  - **Minterm generator** (函数最小项发生器): Generate the 2^n (or fewer) minterms of n input variables. → **Example on page 20 of Lecture 6 & Assignment 2**
  - **Demultiplexer** (数据分配器): A decoder with enable input can function as a **demultiplexer** – a circuit that receives information from a single line and directs it to one of 2^n possible output lines.
  - **Address decoder** (地址解码器): Identify a memory cell, disk sector, or other memory or storage device, to ensure one device can communicate with the processor at one time.

- An **encoder** performs the inverse operation to that of a decoder.
- Example: Octal-to-binary encoder.
  - Eight inputs and three outputs connected with `OR`.
  - Only one input can be active for one time.

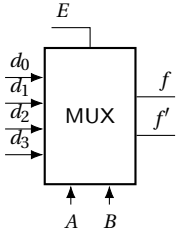
Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

→ Example on pages 27 – 30 of Lecture 6 & Assignment 2

Standard Components (Lecture 6)

Multiplexer (数据选择器/多路复用器)

- A **multiplexer (MUX)** (数据选择器/多路复用器) is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.
- It selects 1-out-of-n lines where n is usually `2`, `4`, `8`, or `16`.
- Example: 4-to-1 MUX
  - A block diagram of a multiplexer having 4 input data lines `d0`, `d1`, `d2`, and `d3` and complementary outputs `f` and `f'`.
  - its characteristic equation is $f = A'B'd_0 + A'Bd_1 + AB'd_2 + ABd_3$.



- Multiplexer as a boolean function generator

→ **Example on pages 36 – 37 of Lecture 6 & Assignment 2**

Exercise

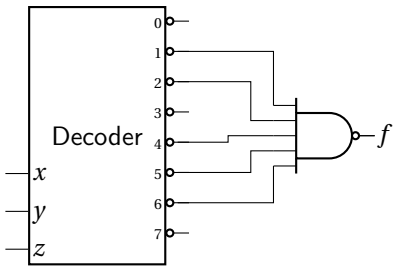
Implement the following Boolean function using a 3-to-8-line decoder: $f(x, y, z) = x'y'z + xy' + yz'$.

Exercise

Implement the following Boolean function using a 3-to-8-line decoder: $f(x,y,z) = x'y'z + xy' + yz'$.

Solution:

$$
\begin{aligned}
f(x,y,z) &= x'y'z + xy' + yz' \\
&= x'y'z + xy'z + xy'z' + xyz' + x'yz' \\
&= \sum(1,5,4,6,2)
\end{aligned}
$$



Exercise

Implement the following Boolean function using a 4-1 multiplexer: $f(x, y, z) = x'y'z + xy' + yz'$. Let x and y be the control signals.

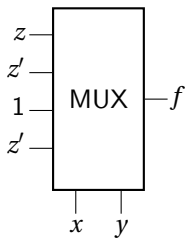
Exercise

Implement the following Boolean function using a 4-1 multiplexer: $f(x,y,z) = x'y'z + xy' + yz'$. Let x and y be the control signals.

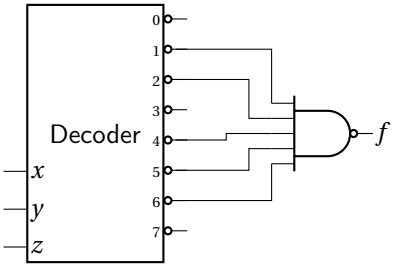
Solution:

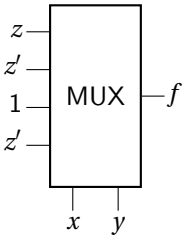
$$f(x,y,z) = x'y'z + xy' + yz' = x'y'z + xy' \cdot 1 + xyz' + x'yz'$$

Decimal	x	y	f
0	0	0	z
1	0	1	z'
2	1	0	1
3	1	1	z'



$$f(x, y, z) = x'y'z + xy' + yz'$$





Encoders and multiplexers can be used to implement any combinatorial circuits.

Arithmetic Circuits (Lecture 7)

- Standard components: combinational circuits that are employed extensively in the design of digital systems.
  - Comparators, decoders, encoders, multiplexers, adders, subtractors and multipliers.
- Adders can be used to implement subtractors and multipliers.
  - The subtraction $X - Y$ can be done by taking the 2's complement of Y and adding it to A. Thus, $X - Y = X + 2's\ complement\ of\ Y$ (Subtraction → Addition)
  - The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits.
  - The 1's complement can be implemented with inverters, and a 1 can be added to the sum through the input carry.

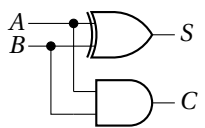
$B_1B_0 \times A_1A_0$

		B_1	B_0
		A_1	A_0
		A_0B_1	A_0B_0
	A_1B_1	A_1B_0	
C_3	C_2	C_1	C_0

Half Adder (Lecture 7)

- A combinational circuit that performs the addition of two bits is called a **half adder**.
- Design of half adder:
  - The circuit needs two binary inputs and two binary outputs.
    - Input variables: the augend (被加数) bit A and addend (加数) bit B.
    - Output variables: the sum S and carry C.
  - The simplified SOP expressions can be obtained directly from the truth table: $S = A \oplus B$ and $C = AB$.
  - Logic diagram can be implemented in SOPs.

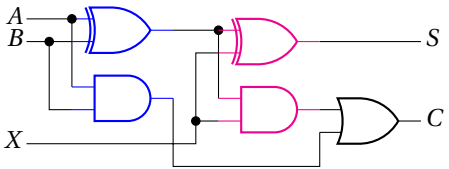
Input		Output	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Full Adder (Lecture 7)

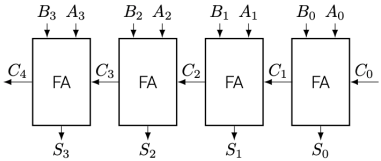
- Addition of n-bit binary numbers requires the use of a full adder.
- A **full adder** is a combinational circuit that forms the arithmetic sum of three bits
  - Input variables:
    - A and B: the two significant bits to be added;
    - X: the carry from the previous lower significant position.
  - Output variables: sum S and carry C.

Input			Output	
A	B	X	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Binary Adder (Lecture 7)

- A **binary adder** is a digital circuit that produces the arithmetic sum of two binary numbers.
- It can be constructed with full adders connected in cascade, i.e., **ripple-carry-adder** (串行进位加法器), with the output carry from each full adder connected to the input carry of the next full adder in the chain.
- Addition of n-bit numbers requires a chain of n full adders or a chain of 1 half adder and $n-1$ full adders.



The 4-bit adder is a typical example of a standard component.

Exercise

Design a binary multiplier that multiplies a 4-bit number, $B_3B_2B_1B_0$, by a 3-bit number $A_2A_1A_0$. The circuit is to be implemented using `AND` gates and `full adders`.

Exercise

Design a binary multiplier that multiplies a 4-bit number, $B_3B_2B_1B_0$, by a 3-bit number $A_2A_1A_0$. The circuit is to be implemented using `AND` gates and `full adders`.

Solution:

"*For J multiplier bits and K multiplicand bits, we need $J \times K$ `AND` gates and $J-1$ K-bit adders to produce a product of $(J+K)$ bits.*" [1]

- There are 3 multiplier bits and 4 multiplicand bits, hence $K=4$ and $J=3$.
- Twelve `AND` gates and two `4-bit adders` are needed to produce a product of 7 bits.

[1] M. M. Mano and M. Ciletti, *Digital design: with an introduction to the Verilog HDL*. Pearson, 2013

"*For J multiplier bits and K multiplicand bits, we need $J \times K$ AND gates and $J-1$ K-bit adders to produce a product of $(J+K)$ bits.*" [1]

- There are 3 multiplier bits and 4 multiplicand bits, hence $K=4$ and $J=3$.
- Twelve `AND` gates and two `4-bit adders` are needed to produce a product of 7 bits.

[1] M. M. Mano and M. Ciletti, *Digital design: with an introduction to the Verilog HDL*. Pearson, 2013

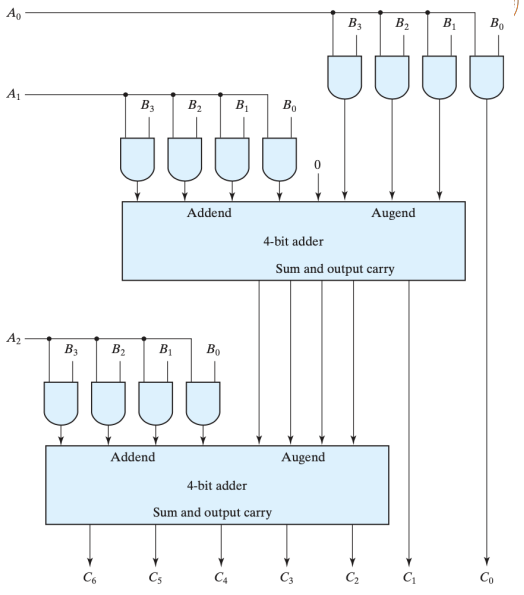


FIGURE 4.16
Four-bit by three-bit binary multiplier

Sequential Circuits (Lectures 8 – 11)

- Sequential circuit: specified by a time sequence of inputs, outputs, and internal states.
- A combinational circuit, but with **memory elements** (记忆元件) connected in a feedback path (反馈回路).
  - **Memory elements**: SR latch, Controlled SR latch, Controlled D latch, DFF, JKFF, TFF (Lecture 8)
  - **Procedure of analysing sequential circuits** (Lecture 9)
    - Diagram -> a table for the time sequence of inputs, outputs, and internal states. Or obtain the Boolean expressions that describes the behaviour of the circuit.
  - **Procedure of designing sequential circuits** (Lecture 9)
    - Specification -> a state table and logic diagram.
  - **Register** (Lecture 10)
  - **Counter** (Lecture 11)

Vocabulary

- Synchronous Sequential Circuits (同步时序电路).
- Asynchronous sequential circuit (异步时序电路).
- clock pulse (时钟脉冲), denoted by `clock` or `clk`.
- Level sensitive device. (时钟信号电平敏感的存储单元电路).
- Edge-sensitive devices (时钟信号边沿敏感的存储单元电路).
- Memory elements (记忆元件):
  - Flip-flop (触发器) (FF).
  - Latch (锁存器).
- Set: S (置位), usually means “set the state set to 1”.
- Reset: R (复位), usually means “set the state to 0”.
- Present state: Q_t or Q (现态).
- Next State: Q_{t+1} or Q' (次态).
- Ways to describe a sequential circuit:
  K-map (卡诺图), logic diagram (逻辑电路图), function table (功能表), state table (状态表), state diagram (状态图), characteristic equations (特性方程) (next state equation, 次态方程), excitation table (激励方程).

Characteristic Tables & Characteristic Equations

- A **characteristic table** describes the logical properties of a flip-flop by describing its operation in tabular form.
- A **characteristic equation** describes the logical properties of a flip-flop by describing its Boolean function.

- DFF: $Q_{t+1} = D$.
- JKFF: $Q_{t+1} = JQ'_t + K'Q_t$.
- TFF: $Q_{t+1} = T \oplus Q_t$.

J	K	Q_{t+1}	
0	0	Q_t	No change
0	1	0	Reset
1	0	1	Set
1	1	Q'_t	Complement

D	Q_{t+1}	
0	0	Reset
1	1	Set

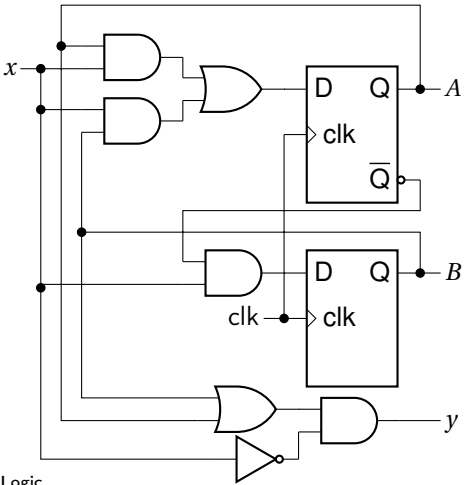
T	Q_{t+1}	
0	Q_t	No change
1	Q'_t	Complement

State Equations or Transition Equations

- The behaviour of a sequential circuit can be described with **state equation**s, or **transition equation**s.

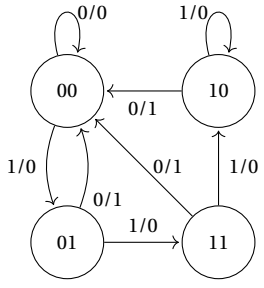
- Two D flip-flops: according to $Q_{t+1} = D$,
  - $A_{t+1} = A_t x_t + B_t x_t$,
  - $B_{t+1} = A'_t x_t$.

- The output:
  - $y_t = [A_t + B_t] x'_t$.
  - Since all signals are labeled by t, we can also write $y = (A + B)x'$.



State Diagrams & State Tables

- Time-sequence of inputs, outputs, FFs of a sequential circuit can be enumerated in a **state table**.
- **State diagram**:
  - Each state as a circle.
  - Transitions between states are directed lines connecting the circles.



Present		Next				Output	
		$x=0$		$x=1$		$x=0$	$x=1$
A	B	A	B	A	B	y	y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

Excitation Tables

Q_t	Q_{t+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Present		Input	Next		Flip-flip Inputs			
A	B	x	A	B	J_A	K_A	J_B	K_B
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	1	0	1	X	X	1
0	1	1	0	1	0	X	X	0
1	0	0	1	0	X	0	0	X
1	0	1	1	1	X	0	1	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1

Analysis of Sequential Circuits (Lecture 9)

- Circuit diagram → Equations → State table → State diagram.
  - State table is easier to derived from circuit diagram and state equations.
  - State diagram gives a pictorial view of state transitions.

- Procedure:
  - Equations: Express algebraically the logic diagram of the circuit with FF input equations and output equation(s).
  - State table:
    - Determining input equations.
    - Listing binary values for each input equation.
    - Using the corresponding flip-flop characteristic table to determine next state values.
  - State diagram: Easy with state table!

→ Example on pages 16 – 21 of Lecture 9.

Design of Sequential Circuits (Lecture 9)

- Given a set of specifications:
  - Different from combinational circuits, a sequential circuit requires a state table.
- It consists of choosing the flip-flops and combinational gates.
  - Number of flip-flops determined from the number of states.
  - Combinational circuits derived from state table.
- Procedure for designing synchronous sequential circuits:
  1. From the word description and specifications of the desired operation, derive a state diagram for the circuit.
  2. Reduce the number of states if necessary. **(Example on pages 26 – 33 of Lecture 9)**
  3. Assign binary values to the states.
  4. Obtain the binary-coded state table.
  5. Choose the type of flip-flops to be used.
  6. Derive the simplified flip-flop input equations and output equations.
  7. Draw the logic diagram.

→ Example on pages 40 – 44 of Lecture 9.

Recap: Question 2 in Assignment 3

Exercise

A sequential circuit has two JK flip-flops A and B and one input x. The circuit is described by the following flip-flop input equations: $J_A = x'$, $K_A = B$, $J_B = x$, $K_B = A'$.

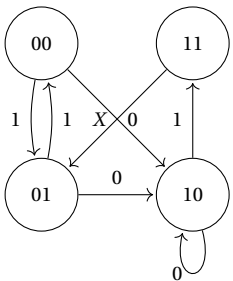
1. Derive the state equations A_{t+1} and B_{t+1} by substituting the input equations for the J and K variables.
2. Draw the state diagram of the circuit.

Solution:

1. According to the characteristic equation of JK FF $Q_{t+1} = JQ'_t + K'Q_t$:

$$
\begin{aligned}
A_{t+1} &= J_A A' + K'_A A = x'A' + B'A \\
B_{t+1} &= J_B B' + K'_B B = xB' + AB
\end{aligned}
$$

2. State diagram of the circuit:



Present		Input	Next	
A	B	x	A	B
0	0	0	1	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	1	1
1	1	0	0	1
1	1	1	0	1

Commonly Used Circuits That Include Flip-flops (Lectures 10 & 11)

- **Register**:
  - A group of flip-flops, each one of which
    - shares a common clock and
    - is capable of storing one bit of binary information.
  - A flip-flop stores one bit of binary information.
    → An n-bit register consists of a group of n flip-flops capable of storing n bits of binary information.
  - In addition to the flip-flops, a register may have combinational gates that perform certain data-processing tasks.

- **Counter**:
  - A special type of register that goes through a predetermined sequence of binary states.

Shift Register (Lectures 10 & 11)

- A register capable of shifting its binary contents either to the left or to the right is called a **shift register** (移位寄存器). → A chain of FFs in cascade (串联).
- The shift register permits the stored data to move from a particular location to some other location within the register.
  - Serial-in serial-out shift register (串入串出移位寄存器).
  - Serial-in parallel-out shift register (串入并出移位寄存器).
  - Parallel-in serial-out shift register (并入串出移位寄存器).
  - Parallel-in parallel-out shift register (并入并出移位寄存器).
- The shift registers can be used to
  - transform between parallel/sequential or sequential/parallel data loading/output;
  - perform arithmetic operations;
  - perform some other data processing operations.

Representative Applications of Shift Register (Lectures 10 & 11)

- Sequence generator (序列信号发生器)
- Serial addition
- Frequency divider
- Counters
  - **ring counter** (环形计数器)
  - **switch-tail ring counter** or **Johnson counter**
  - MOD-N counter
  - (Synchronous / Asynchronous) down counter, up counter, multimode counter

→ **All the examples of Lecture 10 and Lecture 11 are important!**

Exercise

Design a one-input, one-output serial 2's complementer. The circuit accepts a string of bits from the input, and outputs 0's until the first 1 is received. After that, the 2's complement of the input is generated at the output. The circuit can be reset asynchronously to reset the operation.

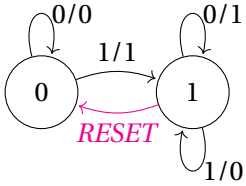
Solution:

- The output is 0 for all 0 inputs until the first 1 occurs at which time, the output is 1. Thereafter, the output is the complement of the input. The state diagram has 2 states:
  - State 0 : Output = Input
  - State 1 : Output = Complement of input
- Note an asynchronous RESET is needed to reset the operation.
- Determine the state table and state diagram.

Present state A_t	Input x	Next state A_{t+1}	Output y
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	0

or

Present state A_t	Next state A_{t+1}		Output y	
	$x=0$	$x=1$	$x=0$	$x=1$
0	0	1	0	1
1	1	1	1	0

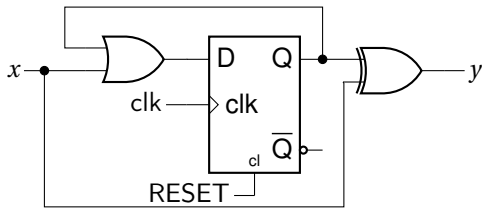


- Determine the excitation equation of y and A_{t+1}:

$$\begin{aligned} y &= A_t'x + A_t x' = A_t \oplus x \\ A_{t+1} &= A_t'x + A_t x' + A_t x = A_t + x \end{aligned}$$

  But, if $RESET = 1$, $y = 0$ and $A_{t+1} = 0$.

- A DFF can be used to implement the circuit.



Lectures 8-11
时序逻辑电路
分析与设计
&
记忆模块元件

Lectures 5-7
组合逻辑电路
分析与设计
&
基本模块元件

Lectures 3 & 4
门级电路
化简与实现

Lectures 1 & 2
基础概念和公理

Sequential circuits

- Key difference compared to combinational circuits: memory elements, feedback path
- Representation of sequential circuits: logic diagram, function table
- Memory elements: latch & flip-flop (key difference: level sensitive devices vs. edge-sensitive devices)
  - (Controlled) SR latch, (Controlled) D latch, DFF (DFF, diagram, function table), JKFF
- State reduction & state assignment
- Analysis and design of sequential circuits (specification -> a logic diagram)

Combinational circuits

- Analysis of combinational circuits: given a diagram, output Boolean functions or truth table
- Design of combinational circuits: specification -> a logic diagram / Boolean functions
- Standard components: encoder (Priority encoder), decoder (minterm generator), multiplexer (Boolean function generator), magnitude comparator, binary adder (carry, half adder, full adder, carry lookahead logic), binary subtractor (borrow, half subtractor, full subtractor, overflow), binary multiplicator (implementation)

Gate-level design

- Gate-level minimisation: Boolean Algebra; Karnaugh map (2/3/4-vars K-maps, don't care)
- Implement circuits: Two/multi-level implementation using NAND, NOR and XOR gates

- Number Systems & Conversions
- Complement of numbers
- Representation of Data -- Code (8421BCD, Gray Code)

- Boolean Algebra & Boolean functions
- SOP/POS forms, sum of minterms/maxterms
- Logic operators and gates