University of Waterloo
Faculty of Engineering
Department of Electrical and Computer Engineering

ECE 124, Digital Circuits and Systems
Winter 2016

J.G. Thistle, EIT 3113, ext. 32910,
jthistle@kingcong.uwaterloo.ca

Calendar description

Number systems and Boolean arithmetic. Boolean algebra and simplification of Boolean functions. Combinational circuits. Sequential circuits; design and implementation. Hardware description languages. Timing analysis. Implementation technologies. [Offered: W, S] Prereq: ECE 140; Level at least 1B Computer Engineering or Electrical Engineering or Software Engineering.

Themes and objectives

This course is an introduction to the building blocks of digital computers, and other digital circuits and systems.

Such systems manipulate information that is encoded in the form of numbers, usually in the base 2, or binary, number system. A pioneer of digital systems engineering, Claude Shannon, observed (in his Master's thesis), that functions defined over the binary numbers could be viewed as terms in the simplest of all Boolean algebras, which is consequently called *switching algebra*. A practical implication is that any such function can be implemented by combining a small number of simple switching elements called *gates*. Standard choices of gates implement logical AND, OR and NOT functions.

Gates usually take the form of electronic circuits, but they can be built using a variety of other technologies. The methods of analysis and design of gate circuits are largely independent of the technology used to implement the gates. This course will not be concerned with electronics, or other implementation technologies, but rather with the analysis and design of schematic gate circuits.

Digital circuits and systems are divided into two main classes: *combinational circuits*, which have no memory and thus no "state"; and *sequential circuits*, which incorporate memory elements and therefore have an internal state whose transitions depend on inputs and on the circuit state itself. We will begin with the study of combinational circuits, which are gate circuits that do not feature any "feedback loops" connecting circuit outputs to inputs. We'll then use feedback to build memory elements such as "latches" and, in particular, "flip-flops" that will allow us to build sequential circuits.

A key theme throughout will be simplification. Often, this will mean reducing the number of gates

used to make up a circuit; in electronic implementations, this reduces the area that a circuit occupies on a chip and the power that it consumes. Another consideration will be speed: every gate introduces some signal-processing delay, so it will sometimes be important to reduce the number of "levels" of logic between inputs and outputs.

Students will learn the key concepts in the lectures and through independent study and assignments. They will then apply those concepts in the lab, electronically implementing both combinational and sequential circuits.

Instructor

J.G. Thistle, EIT 3113, ext. 32910, jthistle@uwaterloo.ca

Lab instructor

Mahdi Elghazali, ext. 31457, melghaza@uwaterloo.ca

Teaching assistants

To be announced on the course's Learn website.

Textbook

M. Morris Mano & Michael D. Ciletti, *Digital Design*, 5th ed., Prentice-Hall, 2007. ISBN-10:0132774208, ISBN-13:9780132774208

Lectures

Lectures for this course are cancelled during midterm week (February 22 - 26). To make up these classes, we shall use the first three scheduled make-up slots, on January 19, February 2, and March 1; the remaining make-up slots may be used in case of emergency.

The last day of classes, Monday, April 4, will be treated as a Friday to make up for the Good Friday holiday.

Evaluation

labs 20%

midterm 25%

final 55%

The midterm is scheduled for Friday, February 26, from 2:30 - 4:20 p.m., at a location to be announced.

Website

Lecture notes, assignments, solutions and announcements will be posted on the course's Learn website, as will information about the labs.

Academic Integrity

Academic integrity is expected in all course activities. If unsure as to what actions constitute academic offences, speak to a member of the teaching staff or consult the website of the Office of Academic Integrity, or the University's Policy 71 – Student Discipline.

Students who believe they have been unjustly penalized for academic offences should consult Policy 70 – Student Petitions and Grievances, and Policy 72 – Student Appeals.

Students with special needs are accommodated with the help of Accessability Services (Needles Hall, room 1132) in such a way that the academic integrity of the course is maintained.

For further information, and links to relevant documents, see
`https://uwaterloo.ca/engineering/current-undergraduate-students/academic-support/course-responsibilities`.

1.2: BINARY NUMBERS

- Strings of binary digits (0's and 1's) are used to represent information
- "Positional" system - digits in various positions representing different powers of a fixed base, called a radix.
- The radix will usually be included as a suffix: $(203)_{10}$ or $(1101)_2$.
- The term "binary digit" is often abbreviated as a bit.

Binary Arithmetic

- When adding two numbers, add digits in the same places and carry as necessary.
- 1 + 1 results in 0 with carry of 1.

"augend" 1 0 1 0 1 1
"addend" + 1 0 0 1 0 1
"sum" 1 0 1 0 0 0 0

- When subtracting, the same rules apply. Borrow from next if necessary.
- 0 - 1 results in 1 with a borrow of 1.

1 0 1 0 1 1 "minuend"
- 1 0 0 1 0 1 "subtrahend"
0 0 0 1 1 0 "difference"

- When multiplying, write out multiplicand for every "1" in the multiplier, moving to that respective digit.
- The sum of all the resulting "partial products" will be the actual product.

"multiplicand" 1 0 1 1
"multiplier" × 0 1 0 1
"partial products" 1 0 1 1
+ 1 0 1 1
"product" 1 1 0 1 1 1

- One may also divide in binary, however, like decimal, it may not terminate.
- Note that when multiplying a binary number by 2, the decimal point is shifted one to the right

1.3: BASE CONVERSIONS

- Suppose we wanted to convert $(78)_{10}$ to binary.
- We can say $\cdots a_3 a_2 a_1 a_0 = (78)_{10}$.
- As the decimal number is even, a_0 must equal 0.
- Dividing both sides by 2, we get $\cdots a_3 a_2 a_1 = (39)_{10}$
- Repeat process, subtracting each a from both sides.
- The binary representation of $(78)_{10}$ is $(1001110)_2$.

(78) ÷ 2	0
(39 - 1) ÷ 2	1
(19 - 1) ÷ 2	1
(9 - 1) ÷ 2	1
(4) ÷ 2	0
(2) ÷ 2	0
1	1

A more general approach (converting to base r)

(1) Divide number by r to set a quotient and remainder (<r)

(2) Remainder becomes least significant digit in base-r representation.

(3) Repeat (1) and (2) with quotient

For fractions, do the opposite.

(i) Multiply the number by r to set an integer part and fractional part (<1)

(ii) integer part becomes most significant digit of base-r representation.

(iii) Repeat procedure with fractional part. It may not terminate.

1.4 : OCTAL AND HEXADECIMAL NUMBERS

- Base-8 (octal) and base-16 (hexa) are often used to shorten binary representations since the bases themselves are powers of 2.
- For conversion from binary, seperate bits into groups of 3 (octal) or 4 (hexa) and write down their decimal equivalents.

$$(10\ 110\ 001\ 101\ 011.111\ 100\ 000\ 110)_2$$
$$= (2\ 6\ 1\ 5\ 3.7\ 4\ 0\ 6)_8$$

$$(10\ 1100\ 0110\ 1011.1111\ 0010)_2$$
$$= (2\ C\ 6\ B.F\ 2)_{16}$$

1.5 : Complements

- Suppose we want to subtract an n-digit subtrahend N from an n-digit minuend M. (M-N)
- Consider: $M + (2^n - N)$ $= 2^n + (M-N)$ $2^n - N$ is called the 2's complement of N.
- If $M \geq N$, the result is an (n+1)-digit number, an end carry is generated.
- Discarding the end carry is equivalent to subtracting 2^n, $2^n + (M-N) - 2^n$
- We are left with M-N, the desired difference.
- If $N > M$, the result is less than 2^n and there is no end carry.
- The result is: $2^n + (M-N)$ $= 2^n - (N-M)$
- This is the two's complement of N-M, the negative equivalent of M-N.

- A two's complement is one larger than the one's complement: $2^n - 1 - N$
- One's complement is obtained by switching 0's and 1's in a number.

Example: Calculate $(1010100)_2 - (1000011)_2$ using 2's complements.

```
  1 0 1 0 1 00
+ 0 1 1 1 1 0 1
-------------------
1 0 0 | 1 0 0 0 1 |
```

2's complement of 1000011.

end carry present, indicating correct answer.

Example: Calculate $(1000011)_2 - (1010100)_2$ using 2's complements.

```
  1 0 0 0 0 1 1
+ 0 1 0 1 1 0 0
---------------
  1 1 0 1 1 1 1
```

2's complement of 1010100.

no end carry. This is the two's complement of answer 1.

- Subtraction can also be carried out using 1's complement.
- If carry bit present, $M > N$, add carried bit to answer.
- If $M \leq N$, result is one's complement of $N - M$.

Example: Perform the two operations above using one's complements.

```
  1 0 1 0 1 00
+ 0 1 1 1 1 0 0
---------------
(1) 0 0 1 0 0 0 0
+               1
---------------
  0 0 1 0 0 0 1
```

```
  1 0 0 0 0 1 1
+ 0 1 0 1 0 1 1
---------------
  1 1 0 1 1 1 0
```

No end carry indicating result is one's complement of $|Y - X|$ or $X - Y$.

- Operation of adding the end digit to the sum is called end-around carry
- We can generate the 2's complement to the r's complement or radix complement
- We can generalize the one's complement to the (r-1)'s complement - diminished radix complement
- The diminished radix complement is obtained by subtracting each digit from r-1.

1.6 : SIGNED BINARY NUMBERS

- Leftmost digit can be used to represent sign:

  0 = + 1 = −

- Singed 2's complement allows representations from -2^{n-1} to $+2^{n-1} - 1$.
- Signed 2's complement numbers can be added in the same way as unsigned, treating the sign bit as a normal bit. (Discarding end carries as usual).

- Result of addition is the signed two's complement of sum - only if sum lies within the range able to be represented by that many bits.
- If not, an overflow has occured.
- Overflow has occured iff the addend and augend have the same sign, but the sum has an opposite sign.

Example: addition of signed numbers.

$$\begin{array}{rl} -6 & 11111010 \\ -13 + & 11110011 \\ \hline -19 \not= & 11101101 \end{array}$$

$$\begin{array}{rl} +64 & 01000000 \\ +64 + & 01000000 \\ \hline & 10000000 \end{array}$$ overflow!

- We can subtract signed numbers by adding to the minuend the negative of the subtrahend.
- We can find the negative of a signed number by taking its 2's complement.

1.7 : OTHER BINARY CODES

- Binary-coded decimal (BCD): encodes each decimal digit (0-9) as a string of four bits: $(243)_{10} = (0010\ 0100\ 0011)_{BCD} = (11110011)_2$.

Example: Perform the following additions in BCD.

$$\begin{array}{rl} 4 & 0100 \\ 8 + & 1000 \\ \hline & 1100 \\ + & 0110 \\ \hline & 00010010 \end{array}$$

$$\begin{array}{rl} & \overset{1}{0}100\ 0100 \\ + & 0101\ 1000 \\ \hline & 1010\ 1100 \\ + & 0110\ +0110 \\ \hline & (0001\ 0000\ 0010)_{BCD} = (102)_{10} \end{array}$$

- When sum does not represent any binary rep. of a number, add $(6)_{10}$ or (
- For positive numbers, we add a "digit" of "0000" to indicate positive.
- $(1001)_{BCD} = (9)_{10}$ are added to the left of negative numbers.

Example: Decimal subtraction:

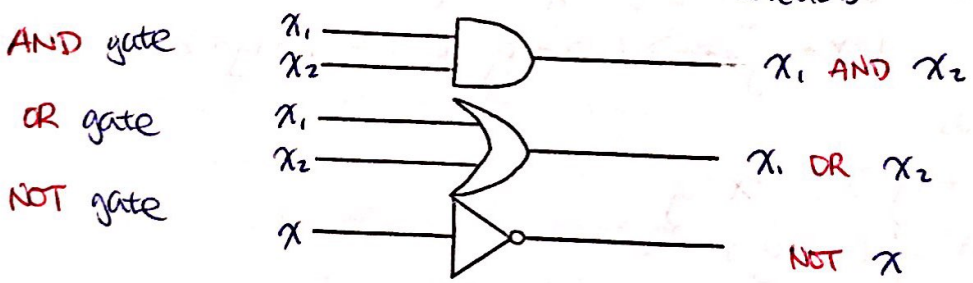
$$\begin{array}{rl} & 24 \\ + & (-22) \\ \hline & 2 \end{array} \quad \Rightarrow \quad \begin{array}{rl} & 024 \\ + & 978 \\ \hline & 1002 \end{array}$$ Discard end-carry.

- In many applications, it's useful to use a code such that the next bit-combination can be obtained by changing only one bit.
- Such a code is known as Gray Code
- Corruption of a single bit (1 changed to 0 or vice-versa) can be detected if a parity bit is added to each word or sequence of bits.
- For even parity, the parity bit is made 0 or 1 so that there are even numbers of
- For odd parity, the parity bit is chosen to make the total number of 1's odd.
- If the parity bit changes, then an odd number of bits are corrupted.
- Such code is called error-detecting code

CHAPTER 2: BOOLEAN ALGEBRA AND LOGIC GATES

Combinational circuits:

- Current outputs solely depend on current inputs
- If we can implement the functions AND, OR, and NOT, then we can implement any binary-valued function.
- Circuit components called gates implement these functions



- Gates are often called logic gates, and circuits composed of them logic circuits.

Boolean Algebra Properties:

(i) $x + 0 = x$; $x \times 1 = x$

(ii) Commutativity: $x + y = y + x$ $\quad x \cdot y = y \cdot x$

(iii) Distributivity: $x + (y \cdot z) = (x + y) \cdot (x + z)$

$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$

(iv) Complement properties: $x + x' = 1$ $\quad x \cdot x' = 0$

2.4: BASIC THEOREMS AND PROPERTIES OF BOOLEAN ALGEBRA

1. Idempotence:

$$x + x = x$$
$$= (x+x) \cdot 1$$
$$= (x+x) \cdot (x+x')$$
$$= x + (x \cdot x')$$
$$= x$$

$$x \cdot x = x$$
$$= x \cdot x + 0$$
$$= x \cdot x + x \cdot x'$$
$$= x(x+x')$$
$$= x$$

- Duality is demonstrated: "dual" equations can be obtained by exchanging "+"'s and "x"'s and "0"'s with "1"'s.

2. Involution:

$$(x')' = x$$

3. Absorption:

$$x + x \cdot y = x$$
$$= x(1+y)$$
$$= x$$

$$x \cdot (x+y) = x + xy$$
$$= x(1+y)$$
$$= x$$

4. Associativity: $x + (y+z) = (x+y) + z$

Note: $(x+y+z) \cdot x = x$

$(x+y+z)(x+y+z) = x+y+z$

5. De morgan's Laws:

$$(x+y')' = x'y'$$
$$x'y'(x+y) = 0$$

Truth tables:

x	y	x AND y
0	0	0
0	1	0
1	0	0
1	1	1

x	y	x OR y
0	0	0
0	1	1
1	0	1
1	1	1

x	NOT x
1	0
0	1

2.5 : BOOLEAN FUNCTIONS

x	y	z	F_1	F_2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	0
1	1	1	1	0

$$F_1 = x'y'z + xy'z' + xy'z + xyz' + xyz$$
$$= xy' + xy + xy'z$$
$$= x + x'y'z + xy'z$$
$$= x + y'z$$

$$F_2 = x'y'z + x'yz + xy'z' + xy'z$$
$$= x'z + xy'$$

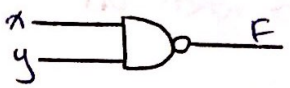
- In expressions such as above, a variable or its complement is a literal.
- A product or sum of literals in called a term.

2.6 : CANONICAL AND STANDARD FORMS

- Suppose we are looking at a boolean function of 3 variables, x, y, and z.
- A minterm is a product of three literals, each with a different variable
- A maxterm is a sum of three literals, each with a different variable.
- A function is equal to the sum of minterms where it equals 1.
- A function is equal to the product of maxterms where it equals 0.
- Minterms can be displayed as m_i, maxterms as M_i.
- Sum of products (minterms) : $f = m_0 + m_2 + m_3 = \Sigma(0,2,3)$
- Product of sums (maxterms) : $f = M_1 + M_4 + M_5 = \Pi(1,4,5)$
- These representations lead to two-level implementations.
- Standard form expressions are unique : $AB + CD + CE$
- Non-standard form expressions are not : $AB + C(D+E)$ ← 3-level.

x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

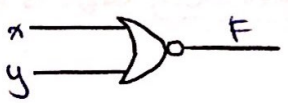
NAND



$F = (xy)'$

x	y	F
0	1	0
0	0	1
1	0	0
1	1	0

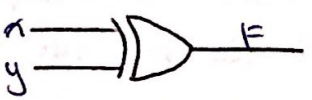
NOR



$F = (x+y)'$

x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

XOR

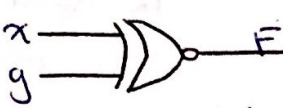


$F = xy' + x'y$

$= x \oplus y$

x	y	F
0	0	1
0	1	0
1	0	
1	1	1

XNOR



$F = xy + x'y'$

$= (x \oplus y)'$

CHAPTER 3: GATE-LEVEL MINIMIZATION

- Two terms are logically adjacent if they differ by only one literal.
- To make minimization more apparent, we use Karnaugh maps (K-maps)
- Note that the row/column on the edges should be logically adjacent.
- Each valid grouping of 1-cells is called an implicant
- A grouping that is not contained in another grouping is a prime implicant
- To get a maximally simplified expression, use only prime implicants.
- If a prime implicant covers one cell that is covered by no other prime implicant, it is called an essential implicant.

3.5: Product of Sums Simplification

- Can be obtained by grouping 0-cells into a sum of products, and applying De Morgan's law.
- A function is incompletely specified if its value is not declared for all input combinations.

3.7: Nand and Nor Implementation

NAND:
$x' = (xx)'$
$xy = ((xy)')'$
$x+y = (x'y')'$

NOR:
$x' = (x+x)'$
$x+y = ((x+y)')'$
$xy = (x'+y')'$

- In general, add inverter bubbles to outputs of AND gates and to the inputs of OR gates. Add inverters or primes as necessary (for nand)
- For nor, add inverter bubbles to outputs of OR gates and to the inputs of AND gates.
- AND-OR-Invert: AND - NOR ≡ NAND - AND
- OR-AND-Invert: OR - NAND ≡ NOR - OR

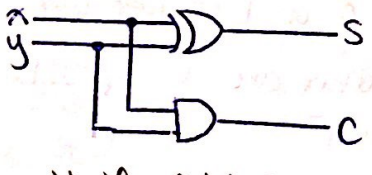
3.8: Exclusive-OR function:

- $x \oplus y$ is only true if exactly one literal is true: $x'y + xy'$
- The K-maps of XOR exhibits a checkerboard pattern.
- The XNOR function is a complement of XOR and is only true when an even number of variables are true.

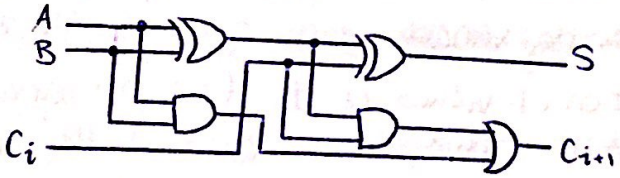
CHAPTER 4: COMBINATIONAL LOGIC

Combinational circuit: a circuit whose outputs are boolean functions of its inputs.

- To analyze a combinational circuit,
  (i) Label all gate outputs and write boolean expressions for them in terms of inputs.
  (ii) Eliminate variables representing intermediate signals
  (iii) Simplify variables.
- A full adder consists of 1-bit addend and augend, a carry input, an output sum bit, and an output carry bit.
- For speed, use two-level implementations from K-maps
- If minimizing the number of gates is important, find similarities between inputs.
- A half adder takes in two inputs and generates a sum bit and carry bit.
- General half adder: $C = xy$, $S = x'y + xy' = x \oplus y$
- 2 half-adders + OR gate = full-adder
- By connecting full adders together, we can create ripple-carry adders
- In an n-bit adder, carries ripple through 2n gates

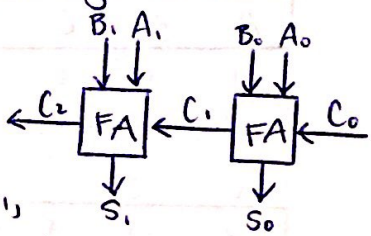


Half-Adder
$C = xy$
$S = x \oplus y$

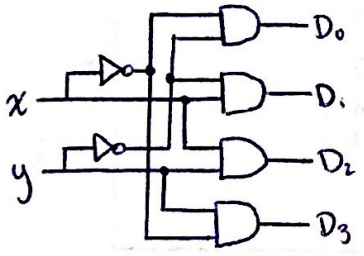


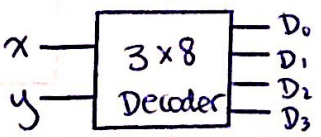
Full-Adder (2 half-adders + OR)
$C = xy + xz + yz = xy + (x \oplus y)z$
$S = x \oplus y \oplus z$

- We can connect n full adders together to create a "n-bit" ripple-carry adder.
- But note that there are 2 levels of logic between C_i and C_{i+1}, so in an n-bit adder, carries have to ripple through 2n logic levels.



4.1: Decoders

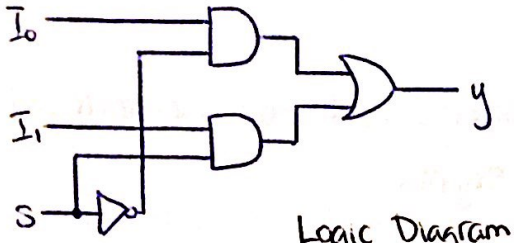




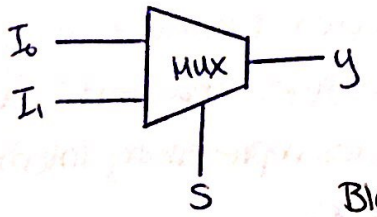
Inputs		Outputs			
x	y	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

4.2 : Multiplexers

- "Switches" that connect selected inputs to their outputs.



Logic Diagram

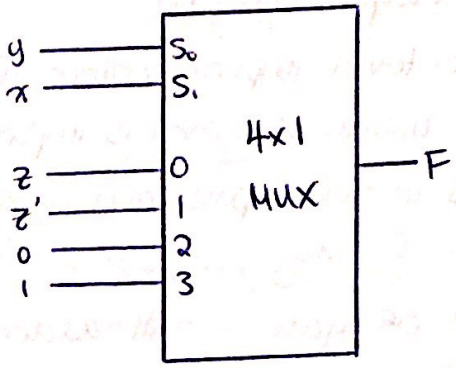


Block Diagram

- We can implement any function with n variables with a 2^{n-1}-to-one line multiplexer.

x	y	z	F	
0	0	0	0	F = z
0	0	1	1	
0	1	0	1	F = z'
0	1	1	0	
1	0	0	0	F = 0
1	0	1	0	
1	1	0	1	F = 1
1	1	1	1	

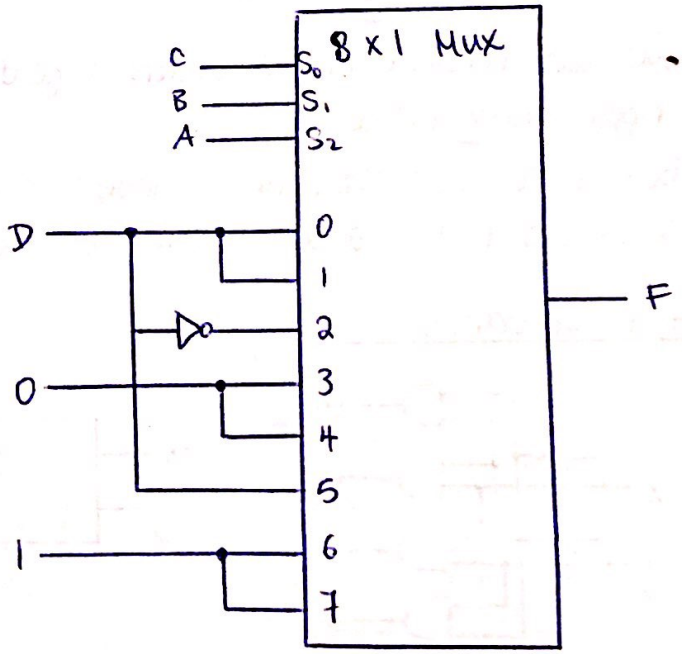
Truth Table



Multiplexer Implementation.

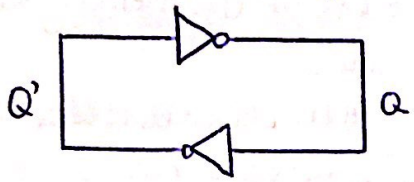
- Let n-1 of the variables be the selector inputs
- If z is the remaining variable, each input line is z, z', 0, or 1 as needed.
- For each combination of values of the first n-1 variables, there are two possible values of z, and two corresponding values of the function, F.
- In both cases, F = z, F = z', F = 0, or F = 1.
- Set the corresponding input to z, z', 0, or 1, accordingly.

A	B	C	D	F	
0	0	0	0	0	F = D
0	0	0	1	1	
0	0	1	0	0	F = D
0	0	1	1	1	
0	1	0	0	1	F = D'
0	1	0	1	0	
0	1	1	0	0	F = 0
0	1	1	1	0	
1	0	0	0	0	F = 0
1	0	0	1	0	
1	0	1	0	0	F = D
1	0	1	1	1	
1	1	0	0	1	F = 1
1	1	0	1	1	
1	1	1	0	1	F = 1
1	1	1	1	1	



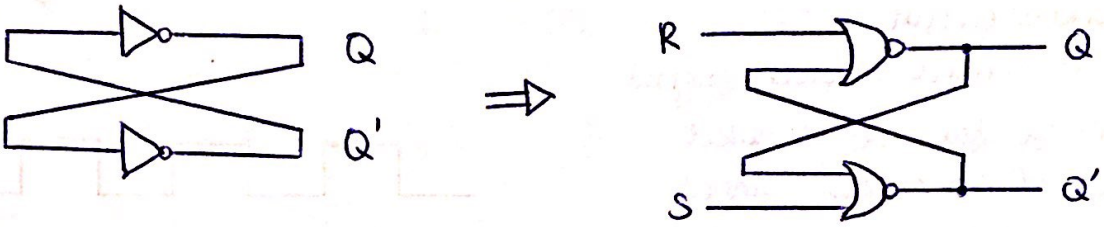
CHAPTER 5: SYNCHRONOUS SEQUENTIAL CIRCUITS

- Let's start by looking at "feedback loops"

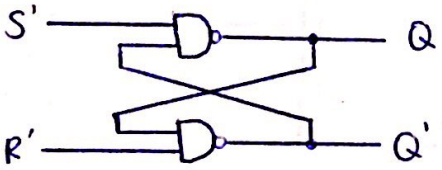


Two possible combinations:
$Q = 0$, $Q' = 1$
$Q = 1$, $Q' = 0$

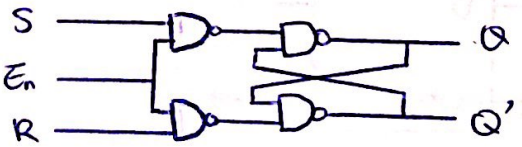
- The circuit has two stable states.
- We can use this circuit to "store" one bit by adding inputs.



- If $S = R$, NOR gates act as inverters:
  → If $S = R = 1$, $Q = Q' = 0$ → If $S = R = 0$, $Q = Q' = 1$.
- If $S = 1$ and $R = 0$, then $Q = 1$ and $Q' = 0$.
  → The circuit is set
- If $S = 0$ and $R = 1$, then $Q = 0$ and $Q' = 1$
  → The circuit is reset.
- Note that if the circuit is set/reset and the inputs are both then set to 0, the values of Q and Q' do not change.
- Since it can store the value of a bit, we call this circuit a latch.

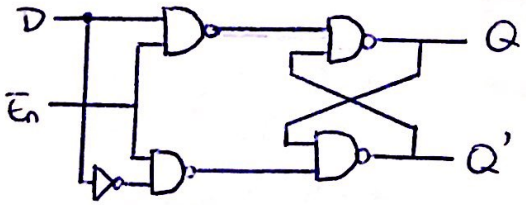


NAND implementation



NAND with Enable Switch.

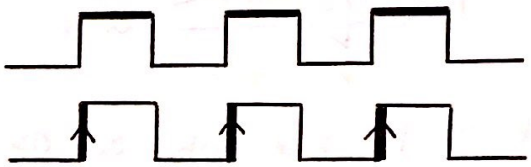
- For last two implementations, next state is $Q = Q' = 1$, invalid.
- Next state becomes indeterminate.
- To avoid this, we use a D-latch:



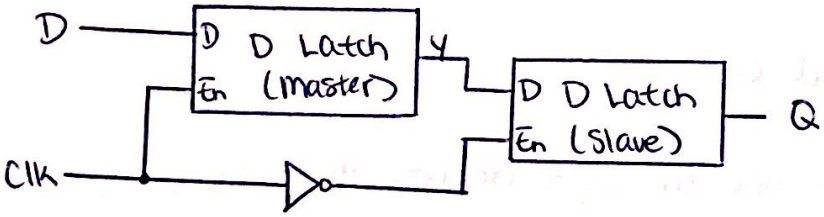
D-latch

Inputs			Current state		Next state		
En	S	R	Q	Q'	Q	Q'	
0	X	X	0	1	0	1	
0	X	X	1	0	1	0	
1	0	0	0	1	0	1	
1	0	0	1	0	1	0	
1	0	1	0	1	0	1	
1	0	1	1	0	0	1	"Reset"
1	1	0	0	1	1	0	
1	1	0	1	0	1	0	"Set"
1	1	1	0	1	1	1	
1	1	1	1	0	1	1	Invalid

- We'll use the above circuits and latches as memory elements
- Circuits containing memory elements are called sequential circuits.
- In a sequential circuit, outputs depend on current states of memory elements
- A change to the input generally provokes a change in the state of the memory element...
- Which can in turn affect the inputs to the memory elements.
- To manage this process, we'd like to enable latches briefly, then allow time for logic outputs to reach their correct values before enabling latches again.
- For this reason, we use "edge-triggered" latches - or flip-flops.
- A sequential circuit in which all memory elements are enabled simultaneously is called a synchronous circuit.
- The enabling signal is called a clock signal.
- Latches we've seen so far are enabled by a positive level of an enabling signal:
- Flip-flops are triggered by either a positive or negative "edge" of the enabling signal:

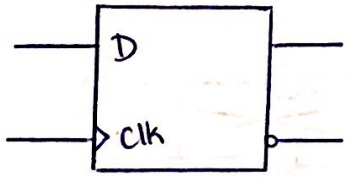


Negative edge-triggered flip-flop:

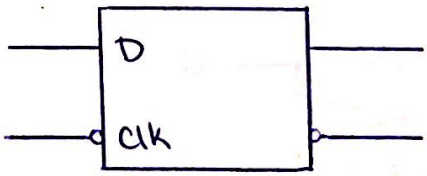


- With clk level high, master is enabled,
- Value of Y tracks that of D
- When clk lowers, master disabled, Y held
- "Slave" is enabled, value of Y is transferred to Q.

Symbols:

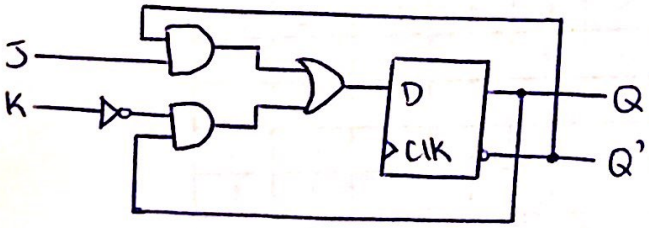


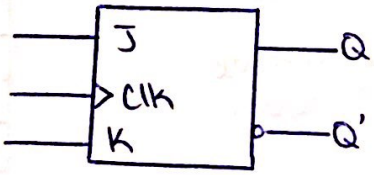
Positive-edge triggering



Negative-edge triggering.

J-K Flip-Flop:





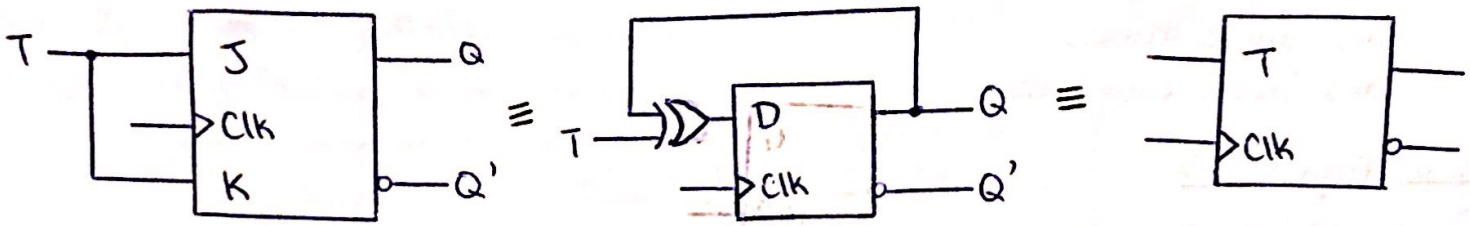
$D = JQ' + K'Q$

$J=1, K=0$
→ $D=1$
→ Flip-flop is set

$J=0, K=1$
→ $D=0$
→ Flip-flop is reset

$J=K=0$
→ $D=Q$
→ Flip-flop held

T- Flip- Flop:



$$D = TQ' + T'Q = T \oplus Q$$

- T- Flip Flops are particularly useful in building counters.

Characteristic Tables

- An alternative to truth-table representation of flip-flops.
- Let Q(t) refer to flip-flop state prior to triggering, Q(t+1) after triggering.

D- Flip- Flops

D	Q(t+1)
0	0
1	1

J-K Flip-Flops

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q'(t)

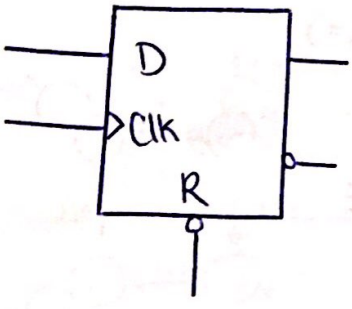
T Flip- Flops

T	Q(t+1)
0	Q(t)
1	Q'(t)

- Q(t+1) = Q(t) : no change. Q(t+1) = Q'(t) : complement
- Q(t+1) = 0 : reset Q(t+1) = 1 : set
- Alternatively, we can write the following characteristic equations:
  - → D Flip-Flop: $Q(t+1) = D$
  - → J-K Flip-Flop: $Q(t+1) = JQ'(t) + K'Q(t)$
  - → T Flip-Flop: $Q(t+1) = TQ'(t) + T'Q(t) = T \oplus Q(t)$

Direct Inputs

- When system is first powered up, states of flip-flops are indeterminate.
- Additional inputs allow the correct initialization of state.
- These are typically asynchronous - they are independent of the clock signal.
- A preset or direct set input sets the flip-flop state to 1.
- A clear or direct reset input sets the flip-flop state to 0.



R	Clk	D	Q	Q'
0	X	X	0	1
1	↑	0	0	1
1	↑	1	1	0

- "R" for direct Reset
- Inverter bubble indicates reset occurs when input is 0.

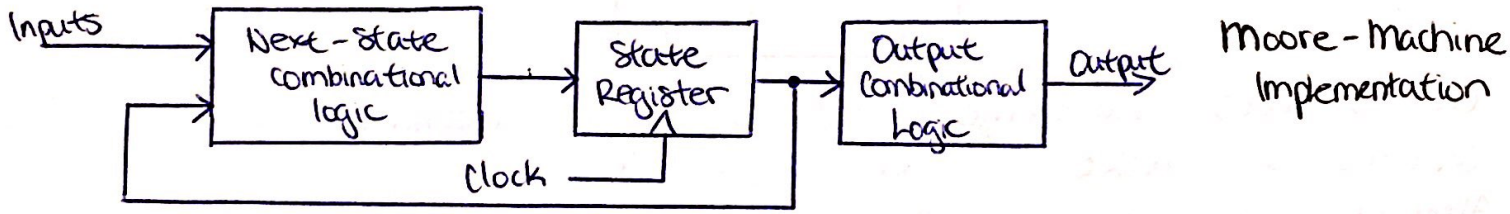
- Analysis of synchronous sequential circuits entails: derivation of:
  (i) State equations
  (ii) State tables
  (iii) State diagrams.

State Equations

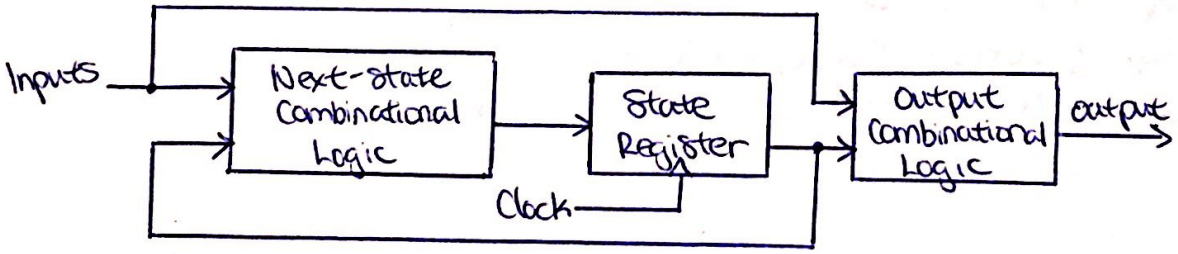
- Gives outputs and next states as functions of current states and inputs.
- Consists of:
  (i) Input equations giving flip-flop inputs as boolean functions of present state and inputs.
  (ii) Characteristic equations of flip-flops
  (iii) Output equations: giving outputs as boolean functions of present state + inputs.

Mealy & Moore Models

- State machines where the output only depends on the present state are called Moore Machines
- State machines whose output depends on both current state and inputs are called Mealy Machines.



Moore-Machine Implementation



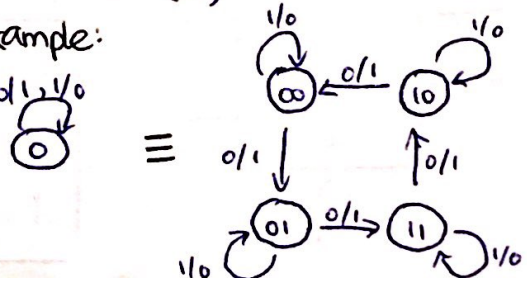
Mealy Machine Implementation

- Produces temporarily incorrect outputs when inputs change, owing to delay before triggering of flip-flops.

State Equivalence

- Two states are equivalent if, for every possible input combination, they give the same output and next states that are equivalent (*)
- The definition is circular, but we resolve the ambiguity by choosing the broadest notion of equivalence that satisfies (*).

Example:



State Assignment

- Once the state diagram (or table) has been established, we need to assign binary strings to the states:

STATE	BINARY	GRAY CODE	ONE-HOT
A	000	000	00001
B	001	001	00010
C	010	011	00100
D	011	010	01000
E	100	110	10000

- Gray-code: useful if the system cycles through its states in sequence - only one flip-flop changes state at a time.
- One-hot: requires as many flip-flops as states, but simplifies next-state and output combinational logic.
- Note that if 2 flip-flops need to change states at once, there will be a glitch on the outputs - the state changes won't happen exactly simultaneously.

Excitation Tables:

- For JK and T flip-flops, we can't get expressions for the flip-flop outputs directly from the state table.
- Instead, we use excitation tables which show how to drive the desired state transitions.

Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

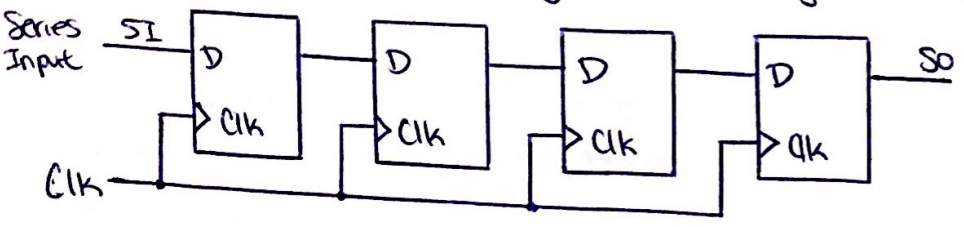
J-K Flip-Flops

Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

T-Flip-Flops

CHAPTER 6: REGISTERS & COUNTERS

- A register is a collection of flip-flops that store a collection of bits.
- Counters are a class of registers that cycle through predetermined state sequences



4-bit Shift register