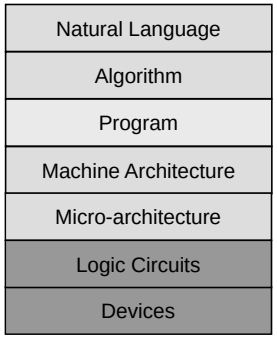
These notes are Dr Delmas' and many other people's notes

Chapter 3

Digital Logic Structures



The Building Blocks of Computers

- Logic gates are the basic building blocks of computers
  - AND, NOT, …
- Combining the logic functions together, higher-level structures can be built
  - Adder, multiplexer, decoder, register, …
- A processor can be built from the circuits providing higher-level functions

Agenda & Reading

- Logic gates
- Simple logic circuits

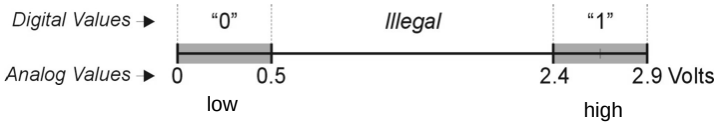
- Read Chapter 3 of the textbook

Agenda

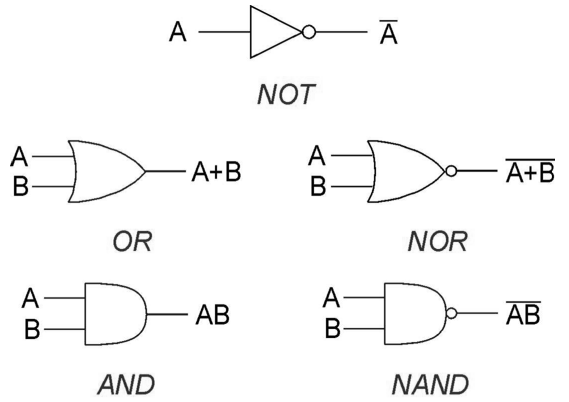
- **Logic gates**
  - **The functionality of logic gates**
  - **How to convert a logic expression into an expression that contains NOT, NOR and NAND operators only**
- Simple logic circuits

Digital symbols

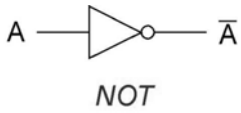
- A range of analog voltages is assigned to represent each of the digital (logic) symbols



Basic Logic Gates

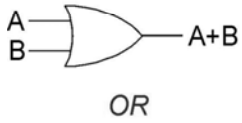


Inverter (NOT Gate)



In	Out
0	1
1	0

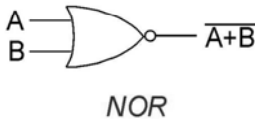
OR Gate



A + B

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

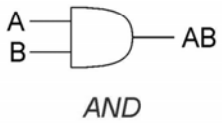
NOR Gate



NOR

A	B	C ← $\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

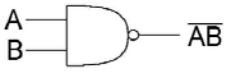
AND Gate



AND

A	B	C → $A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

NAND Gate



NAND

A	B	C → $\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

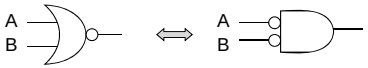
DeMorgan's Law (1)

- NAND, NOR, NOT are the three types of gates for building logic circuits
- When designing a logic circuit, it is more convenient to use AND, OR, NOT in the design.
- Before a design is sent to the factory for fabrication, the design needs to be converted to a design that consists of NAND, NOR, NOT gates only
- The conversion is carried out using the DeMorgan's Law

DeMorgan's Law (2)

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

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



DeMorgan's Law (3)

- To apply the DeMorgan Law to convert a circuit, you can first apply the invert operation twice to the expression that you want to convert

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

Using the DeMorgan Law

- How to implement any gate with two-input NAND (or NOR) gates?

$$A + B = \overline{\overline{A + B}}$$
$$= \overline{\overline{A} \cdot \overline{B}}$$
$$= \overline{\overline{A \cdot 1} \cdot \overline{B \cdot 1}}$$

- Exercise: How to implement a three-input OR gate with two-input NAND and NOR gates?

Agenda

- Logic gates
- **Simple logic circuits**
  - **Build combinational circuits to carry out simple operations, e.g. addition, selection, etc.**
  - **Build sequential circuits to store information, e.g. registers**

Building Functions from Logic Gates

Combinational Logic Circuit
- output depends only on the current inputs
- stateless
- adder, decoder, etc.

Sequential Logic Circuit
- output depends on the sequence of inputs (past and present)
- stores information (state) from past inputs
- register

Combinational Logic Circuit (1)

- The functionality of a combinational logic circuit can be represented using a truth table
- In the table below, A and B are the inputs of the circuit and C is the output of the circuit

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Combinational Logic Circuit (2)

- The functionality of the circuit can be represented in terms of the inputs of the circuit using a logic expression
- The logic expression can be obtained from the truth table as below:
  - Identify the rows with output equal to 1
  - Use AND and NOT gates to make these rows' output equal to 1
  - Use OR gates to combine all the outputs from these rows
- From the logic expression, a logic circuit can be obtained

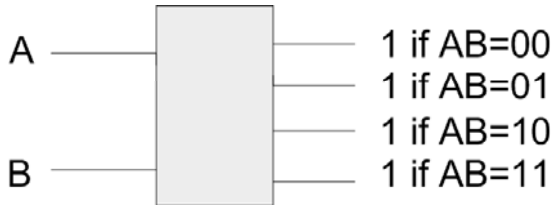
Combinational Logic Circuit (3)

- A and B are the inputs and C is the output
- The second and the third rows' outputs are 1
- How to write an expression written in terms of A and B to make the expression evaluate to 1 when A and B take the value in row 2/3
  - row 2 : (NOT A) AND B
  - row 3 : A AND (NOT B)
- How to write an expression to make C equal to 1 if either row 2 or 3 is 1
  - (row 2) OR (row 3)

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Decoder (1)

- A decoder is used to interpret the input value
  - n input lines, i.e. there are 2^n possible input pattern
  - 2^n output lines
  - exactly one output is 1 for each possible input pattern



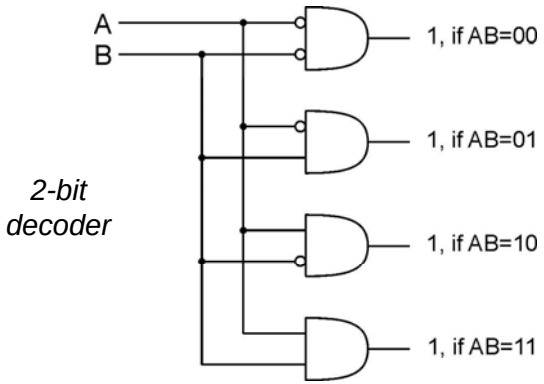
Decoder (2)

- Truth table for output line "if AB=00" is shown
- Only the first row's output is 1
- The expression written in terms of A and B to make the expression evaluate to 1 when A and B take the value in row 1 is
  - (NOT A) AND (NOT B)
- Since row 1 is the only row with output equal to 1, the logic expression corresponds to the truth table is
  - (NOT A) AND (NOT B)

A	B	C
0	**0**	**1**
0	**1**	**0**
1	**0**	**0**
1	**1**	**0**

- The expression for the other output lines can be obtained similarly

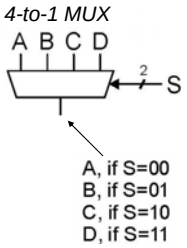
Decoder (3)



2-bit decoder

Multiplexer (MUX) (1)

- A multiplexer can be use to control the information flowing through the system
  - 2^n input data lines,
  - one output data line
  - *n*-bit selector lines
  - output equals to one of the input data lines, depending on the value presented to the selector lines

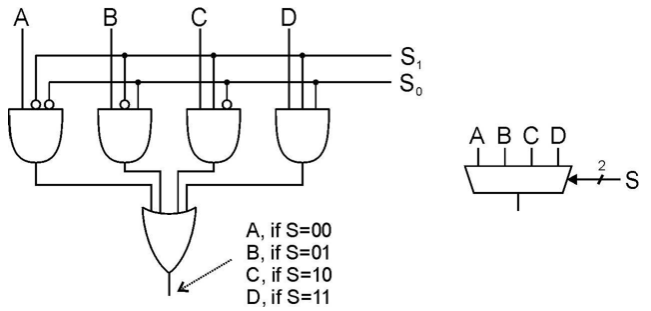


Multiplexer (MUX) (2)

- The AND gate can be used to control the flow of information
  - For an AND gate with two inputs, input and S
    - input can flow through the AND gate if S is 1, i.e. input AND S = input (if S =1)
    - otherwise, input cannot flow through the AND gate
- For the MUX in the previous slide,

$$Out = A \cdot \overline{S}_1 \cdot \overline{S}_0 + B \cdot \overline{S}_1 \cdot S_0 + C \cdot S_1 \cdot \overline{S}_0 + D \cdot S_1 \cdot S_0$$

Multiplexer (MUX) (3)



Half Adder

- A half adder adds two bits together
  - 2 inputs, i.e. the two bits to be added
  - 2 outputs: sum and carry

a_i	b_i	c_{i+1}	s_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Full Adder (1)

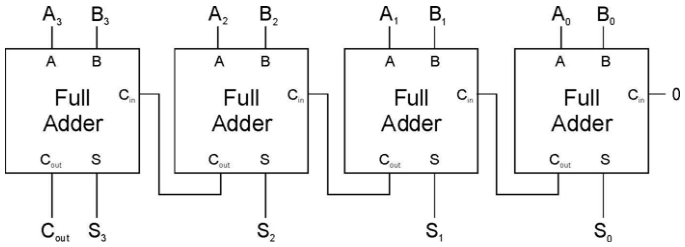
- A number consists of multiple bits.
- When performs addition on bit i, we need to consider the carry in from bit i-1
- A full adder
  - 3 inputs: a_i, b_i and c_i
  - 2 outputs: s_i and c_{i+1}
  - c_i is the carry in from bit position i-1
  - c_{i+1} is the carry out to bit position i+1

$$\begin{array}{r} a_{n-1}\ a_{n-2}\ \ldots\ a_1\ a_0 \\ +\ b_{n-1}\ b_{n-2}\ \ldots\ b_1\ b_0 \\ +\ c_{n-1}\ c_{n-2}\ \ldots\ c_1\ 0 \\ \hline s_{n-1}\ s_{n-2}\ \ldots\ s_1\ s_0 \end{array}$$

Full Adder (2)

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Four-bit Adder

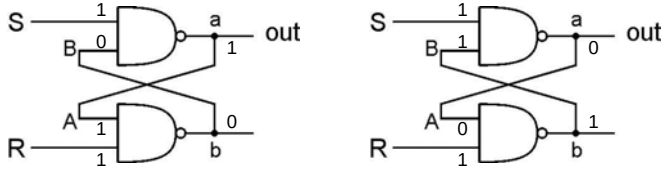


Sequential Circuit

- A sequential circuit can store information
- It is useful for building “memory” elements
- We are going to learn
  - Simple Storage Elements, i.e. RS latch and D latch
  - Building registers and memory

R-S Latch

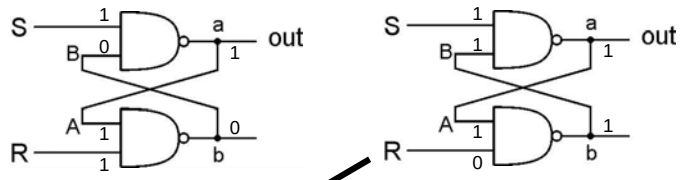
R is used to “reset” or “clear” the element – set it to zero.
S is used to “set” the element – set it to one.



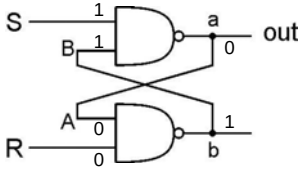
If both R and S are one, out retains the current value.
- “quiescent” state -- holds its previous value
- note: if a is 1, then b is 0, and vice versa

Clearing the R-S latch

Suppose we start with output = 1, then change R to zero.



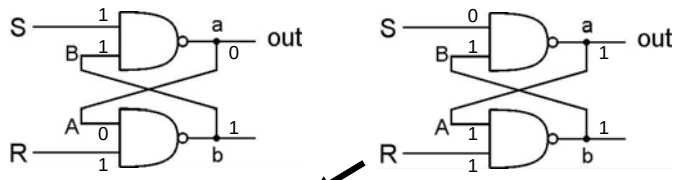
Output changes to zero.



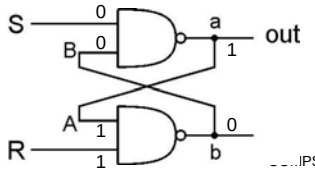
Then set R=1 to "store" value in quiescent state.

Setting the R-S Latch

Suppose we start with output = 0, then change S to zero.



Output changes to one.



Then set S=1 to "store" value in quiescent state.

R-S Latch Summary

R = S = 1
- hold current value in latch

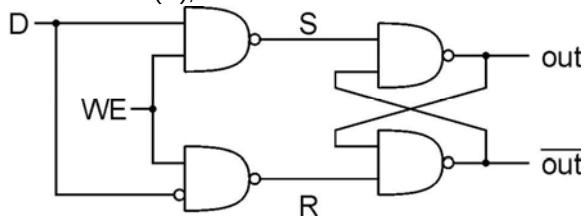
S = 0, R=1
- set value to 1

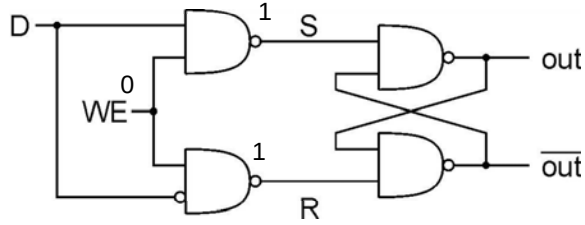
R = 0, S = 1
- set value to 0

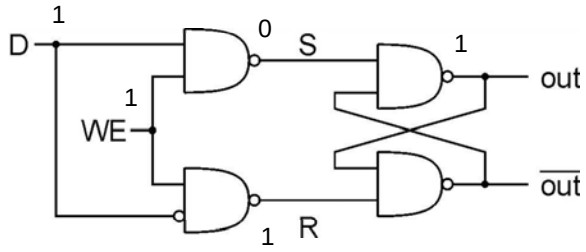
R = S = 0
- both outputs equal one
- final state determined by electrical properties of gates
- ***Don't do it!***

Gated D-Latch

Two inputs: D (data) and WE (write enable)
- when WE = 0, latch holds previous value
  - S = R = 1
- when WE = 1, latch is set to value of D
  - S = NOT(D), R = D



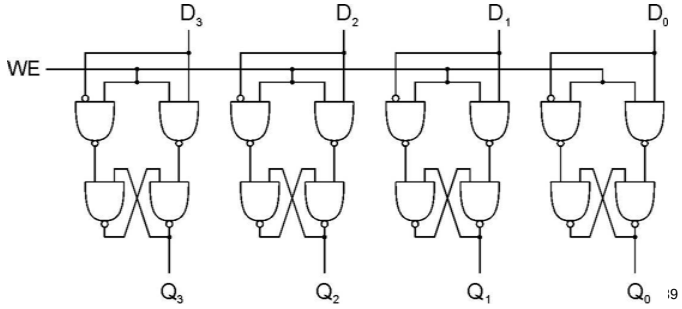




Register

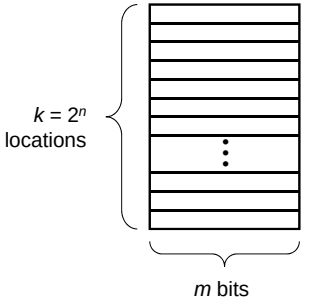
A register stores a multi-bit value.

- We use a collection of D-latches, all controlled by a common WE.
- When WE=1, n-bit value D is written to register.



Memory

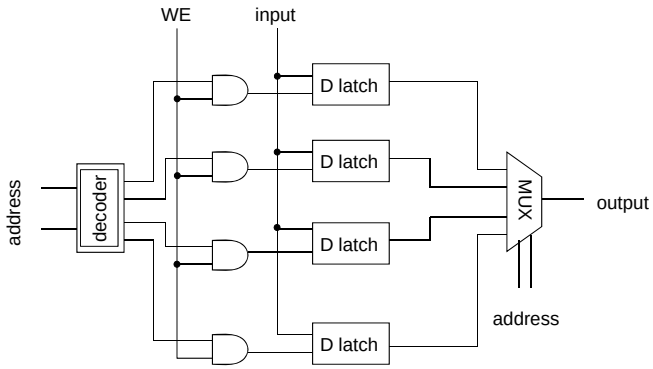
- The memory module consists of a number of locations
- Each location has several bits
- The memory module can be built as a logical $k \times m$ array of stored bits



Building a 4 x 1-bit Memory (1)

- Each location consists of 1 bit
  - Each bit is a gated D-latch
- There are 4 locations
- When read from or write to a location, the address of the location is given
- How to read from a given location
  - MUX:
    - the value of each location is the input of the MUX,
    - the address of the location are the selection line
    - the output of the MUX is the value of the location being read
- How to write to a given location
  - Decoder:
    - the address of the location is the input to the decoder
    - each output line of the decoder connects to the WE line of one of the locations

Building a 4 x 1-bit Memory (2)



reviews

- How the logic gates behave
  - NOT, NAND, NOR, OR, AND
- How to use the DeMorgan's Law to convert a logic expression that contains NAND/NOR operators only
- The differences between combinational and sequential circuits
- Write a logic expression to represent the functionalities specified in a truth table
- RS and D latch
  - How they work
  - Build storage devices using the latches