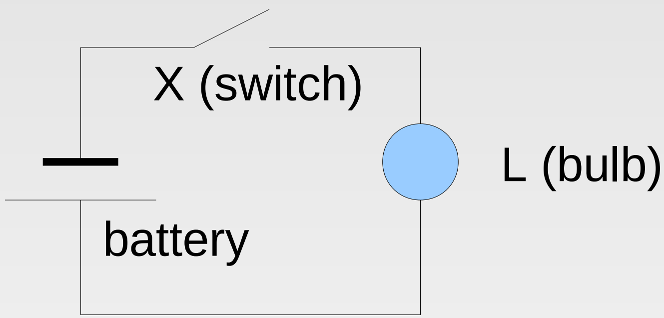
CS2502 – Logic Design

- Logic Design is primarily about **digital circuits,** such as computer hardware.
- LD can also be applied to more general digital systems, such as **computer software**.

CS2502 – 1 Combinational Circuits

1.1 Introduction

Circuit



Observed bahaviour

X - switch	L - bulb
open	off
closed	on

Analysis

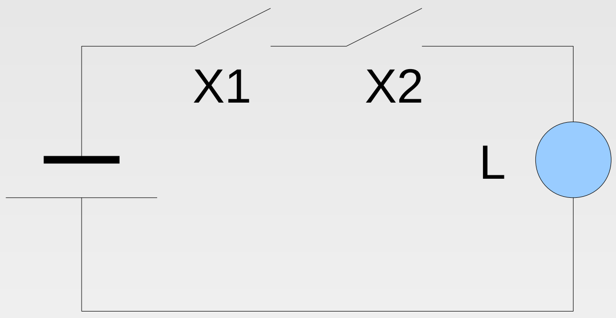
- Q1: How many **states** can the circuit assume?
- Q2: How do X and L **depend** from each other?
- Q3: How can we distinguish the states **more simply**?
- A1: The circuit can assume **two states**.
- A2: L depends on X, but X does not depend on L.
- A3: We can describe the states by 0 and 1.

Description with 0 and 1

X open → 0, X closed → 1
L off → 0, L on → 1

Input X	Output L
0	0
1	1

1.2 AND, OR and NOT



Circuit

X1	X2	L
open	open	off
open	closed	off
closed	open	off
closed	closed	on

X1	X2	L
0	0	0
0	1	0
1	0	0
1	1	1

The AND Operation

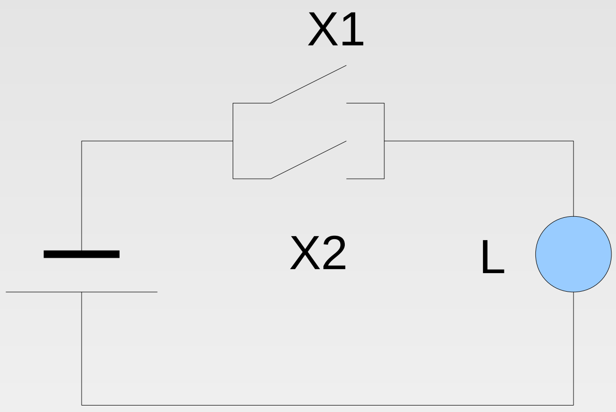
- X1 and X2 are **inputs**.
- L is the only **output**.
- L is on iff X1 **AND** X2 are closed.
- L assumes 1 if both X1 **AND** X2 assume 1. L is 0 otherwise.
- The AND operation is the logic equivalent of the **multiplication**.
- Notations:

L = X1 AND X2

L = X1 ^ X2

L = X1 * X2

A different Circuit



Circuit

X1	X2	L
open	open	off
open	closed	on
closed	open	on
closed	closed	on

X1	X2	L
0	0	0
0	1	1
1	0	1
1	1	1

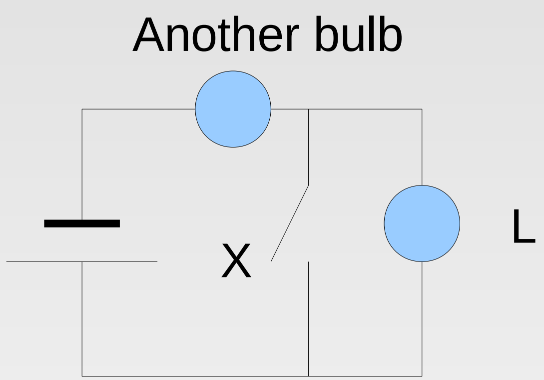
The OR Operation

- X1 and X2 are **inputs**.
- L is the only **output**.
- L is on if X1 **OR** X2 are closed.
- L assumes 1 as soon as X1 **OR** X2 assumes 1.
- The AND operation is the logic equivalent of the **addition**.
- Notations:

L = X1 OR X2
L = X1 V X2
L = X1 + X2

The NOT Operation

Circuit



Observed bahaviour

X - switch	L - bulb
open	on
closed	off

X	L
0	1
1	0

L = NOT X

1.3 Boolean Algebra

Rules

- **Commutative**: A + B = B + A
- A * B = B * A
- **Associative**: A + (B + C) = (A + B) + C
- A * (B * C) = (A * B) * C
- **Distributive:** A * (B + C) = A * B + A * C
- A + B * C = (A + B) * (A + C)

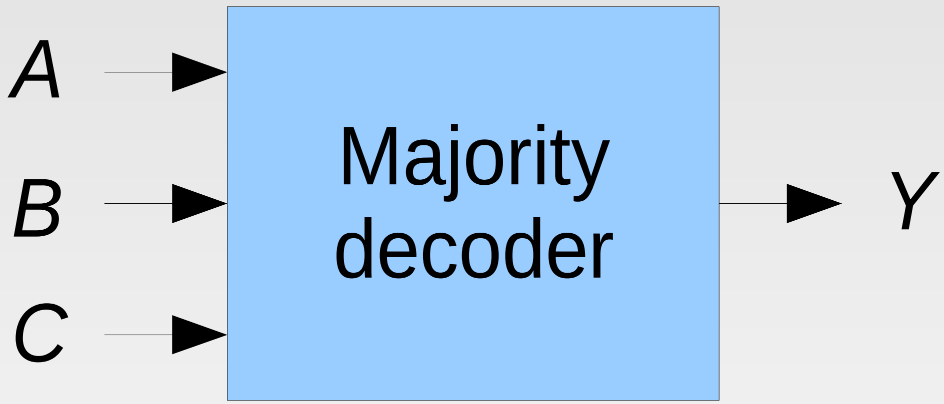
Rules (continued)

- **0 is neutral to +**: $A + 0 = A$
- **1 is neutral to ***: $A * 1 = A$
- **Property of NOT**: $A + NOT(A) = 1$
- $A * NOT(A) = 0$

Boolean Algebra vs Numerical Algebra

- All Values are either **0 or 1**.
- The NOT function is a fundamental operation.
- 1 + 1 = 1
- **Both**, + and * are **distributive** with respect to the other operation.
- Neither + nor * have an inverse operation, i.e. **subtraction** and **division** do **not exist**.

Design Example: Majority Decoder



- 3 inputs: *A*, *B* and *C*
- One output *Y*
- *Y* shall asume that value which is carried by the **majority** of inputs.
- Therefore, *Y* assumes 1 **iff two or all three** inputs are 1.

Applications of Majority Decoders

- Alarm systems. In systems with three sensors, an alarm is raised if **at least two** sensors are activated.
- The, so called, **full adder** is used to add binary numbers. The **carry-over** output of a full adder is the same logic function which is performed by a majority decoder.
- Majority decoders with more than three inputs are possible. However, the number of inputs must **be odd**.

Majority Decoder: Truth Table

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

Majority Decoder: Switching Function

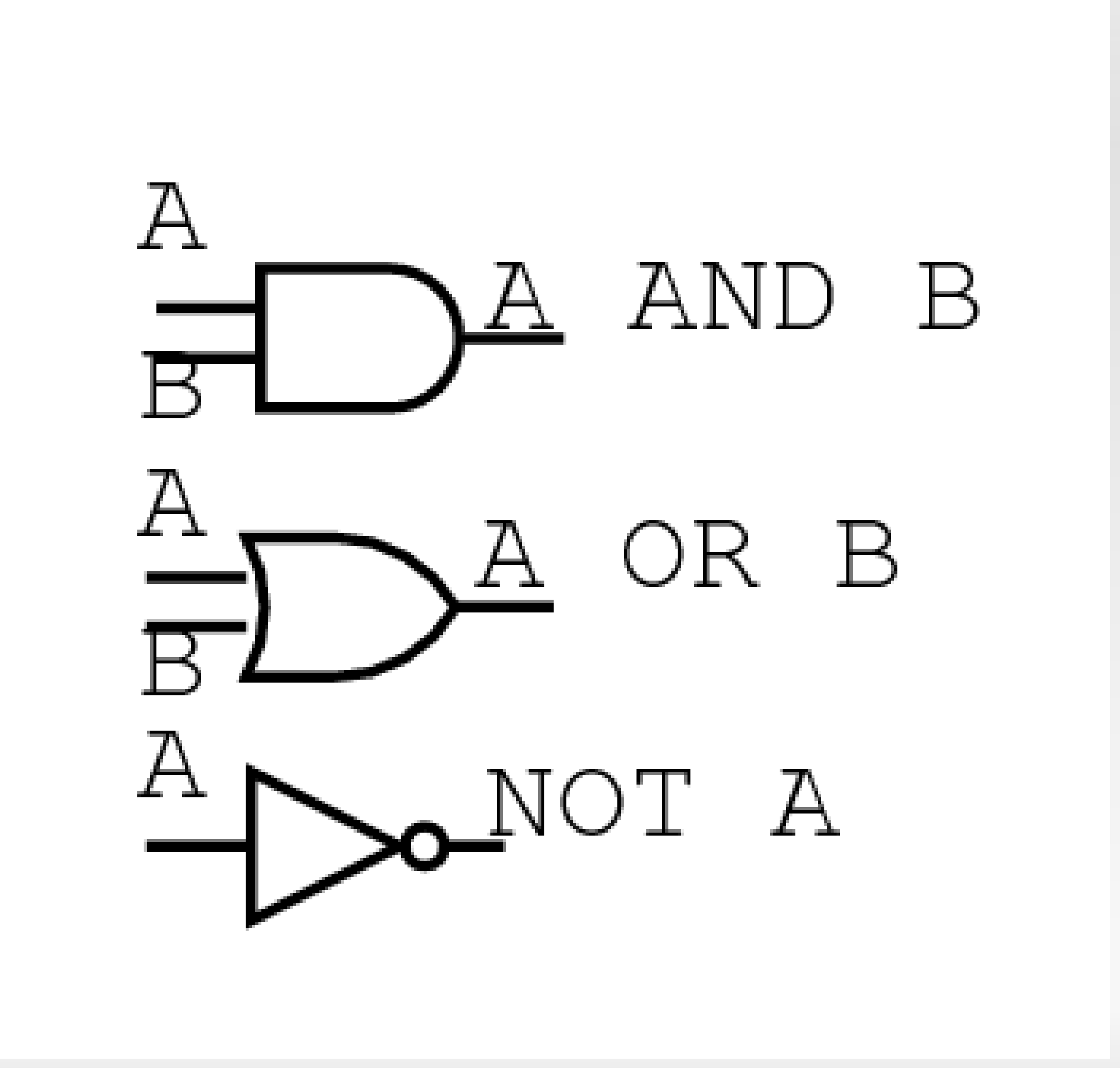
From Truth Table:

$$Y = \neg A \cdot B \cdot C + A \cdot \neg B \cdot C + A \cdot B \cdot \neg C + A \cdot B \cdot C$$

After Simplification: $Y = A \cdot B + B \cdot C + A \cdot C$

(Algorithms for simplification will be taught later in this course.)

1.4 Gates and Circuit Diagrams



1.5 Minterms and Maxterms

Important properties of AND and OR

AND	OR
Assumes **1** iff **all** arguments are **1**.	Assumes **0** iff **all** arguments are **0**.
Assumes **0** if **any** argument is **0**.	Assumes **1** if **any** argument is **1**.

Minterms, Maxterms (continued)

Now, we consider products and sums where some of the arguments are **inverted**.

For example:

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

$$Y = A + \overline{B} + C$$

Minterms, Maxterms (continued)

A	B	C	A*NOT(B)*C	A+NOT(B)+C
0	0	0	0	1
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	1
1	0	1	1	1
1	1	0	0	1
1	1	1	0	1

Minterms, Maxterms (continued)

A **minterm** is a **product of all variables**, some of which might be inverted
A **maxterm** is a **sum of all variables**, some of which might be inverted..

Minterms and maxterms of n variables:

Minterm	Maxterm
Assumes **1** in exactly 1 case.	Assumes **0** in exactly 1 case..
Assumes **0** in 2^n-1 cases.	Assumes **1** in 2^n-1 cases.

1.6 The Canonical Sum of Products

- Any switching function can be expressed as a **canonical sum of products** (CSOP).
- Each **occurrence of 1** in the Y-column of the truth table corresponds to **one minterm**.
- In most cases, the CSOP form is **not the optimal** expression of a switching function.

Short Notations for CSOP Form

Majority Decoder with 3 Inputs:

$$Y = m_3 + m_5 + m_6 + m_7$$

Note that this switching function assumes 1 for those combinations of input values which are listed in row number **3, 5, 6** and **7** in the truth table.

1.7 The Canonical Product of Sums

- Any switching function can be expressed as a **canonical product of sums** (CPOS).
- Each **occurrence of 0** in the Y-column of the truth table corresponds to **one maxterm**.
- In most cases, the CPOS form is **not the optimal** expression of a switching function.

Short Notations for CPOS Form

Majority Decoder with 3 Inputs:

$$Y = M_0 \cdot M_1 \cdot M_2 \cdot M_4$$

Note that this switching function assumes 0 for those combinations of input values which are listed in row number **0, 1, 2** and **4** in the truth table.

Design Example 2: 2-to-1 Multiplexer

Design a circuit with 3 inputs A, B and C and one output Y that meets these specifications:

- If C=0 then Y=A
- If C=1 then Y=B

In other words, depending on the value of C, either A or B will be connected to the output Y.
A and B are also called data inputs and C is also called selection input.

1.8 Simplification of Boolean Expressions

A related example (bank loan or similar):

- *Type 1*: Applicants must be **older than 25** and **married**.
- *Type 2*: Applicants must be **older than 25** and **single**.
- *Type 3*: Applicants must be eligable to **Type 1 or to Type 2.**

Simplifications (continued)

It is obvious that *Type 3* is equivalent to

Applicants must be **older than 25 and married or single**.

Which simply means

Applicants must be **older than 25**.

Simplifications (continued)

Logic equivalent:

$$A \cdot B + A \cdot \bar{B} = A \cdot \left(B + \bar{B} \right) = A$$

This formula is the underlying principle of simplification with **Karnaugh** maps.

Karnaugh Maps

CD

AB	00	01	11	10
00				
01				
11				
10				

Layout of a Karnaugh Map for switching functions of 4 variables.

Karnaugh Maps (continued)

Calculating the
Simplified Sum of Products (SSOP):

- Cover all 1s with **rectangular** blocks of size 1,2,4,8,...
- Try to **maximize the size** of the blocks.
- Try to **minimize the number** of blocks.
- **Overlapping** is allowed.
- Each block corresponds to a **simplified sum**.

Karnaugh Maps (continued)

Calculating the
Simplified Product of Sums (SPOS):

- Cover all 0s with **rectangular** blocks of size 1,2,4,8,...
- Try to **maximize the size** of the blocks.
- Try to **minimize the number** of blocks.
- **Overlapping** is allowed.
- Each block corresponds to a **simplified product**.

Karnaugh Maps with Don't Care Entries

We use this method whenever the switching function is **not fully specified**, i.e. when certain combinations of input values are of **no interest**.

Examples are:

- 2-to-1 multiplexer with **only one** ”clamp” case
- Majority decoders with an **even** number of inputs
- Comparator circuits where the output in case of **equal inputs** is of no interest
- BCD-to-7 segment decoder

Karnaugh-Maps with Don't Care Entries (continued)

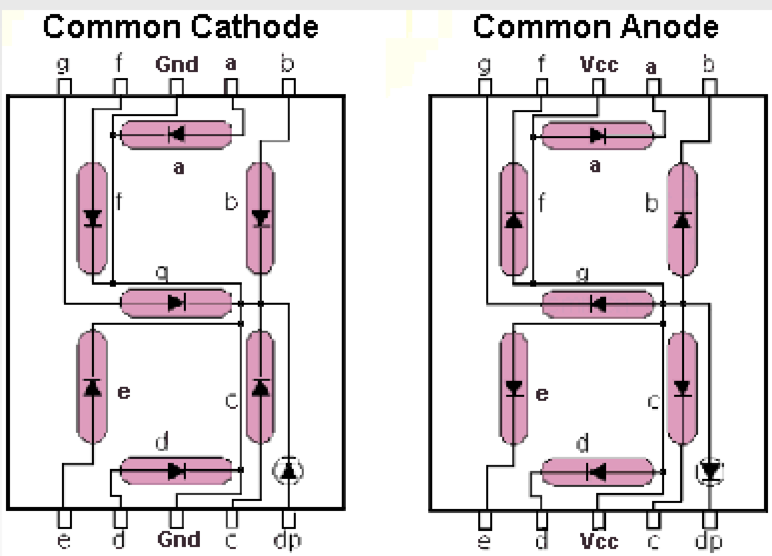
We extend the algorithms:

- In addition to 1 and 0, we use a new entry Ø to indicate that Y might be either 1 or 0.
- Each Ø entry may be **inside or outside** the block coverage.
- M**inimize the numbe**r and **maximize the size** of blocks as before.

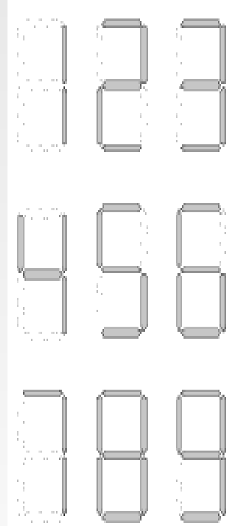
Example:
BCD-to-7 segment Decoder

The complete circuit has 4 inputs and 7 outputs.

7-segment LED display:



Truth Tables of the 7 switching functions:



Digit Shown	a	b	c	d	e	f	g
	Illuminated Segment (1 = illumination)						
0	1	1	1	1	1	1	0
1	0	1	0	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

1.9 De Morgan's Theorem

Relation between minterms and maxterms:

$$m_n = \overline{M_n}$$

This formula can be rewritten in many ways, e.g.:

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

Applying De Morgan's Theoreme

To rewrite a Boolean expression into **NAND-only** form:

- Replace every OR operator with AND
- Invert each **argument** of the former OR operations
- Invert the **result** of every former OR operation
- If the outermost operation is not an inversion, add a **pair of invertions** across the whole expression

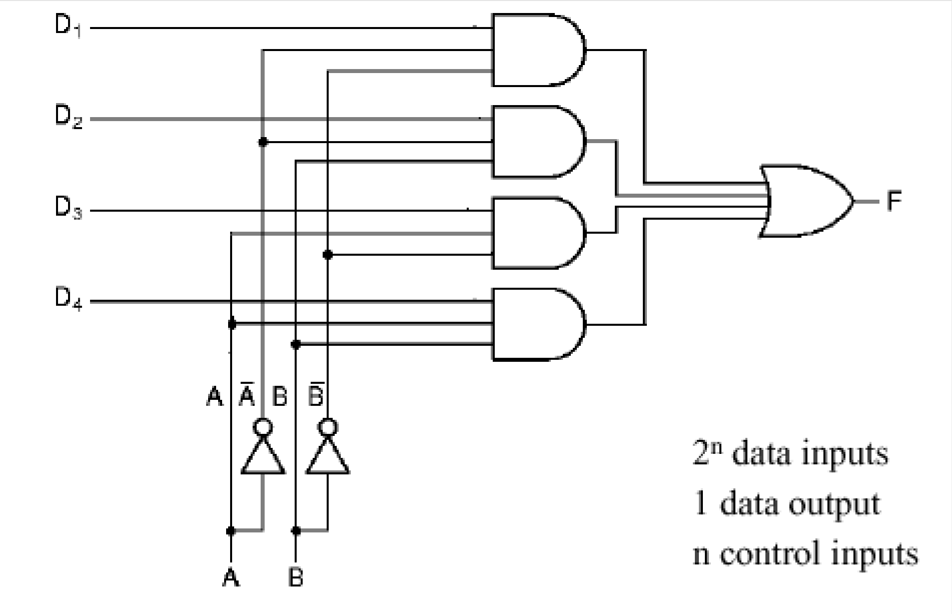
(continued)

To rewrite a Boolean expression into **NOR-only** form:

- Replace every AND operator with OR
- Invert each **argument** of the former AND operations
- Invert the **result** of every former AND operation
- If the outermost operation is not an inversion, add a **pair of invertions** across the whole expression

1.10 Multiplexers

Multiplexer



Depending on A and B, one of the data inputs will be routed to the output.

...continued

Switching function of Multiplexer:

$$Y = D_0 \cdot m_0 + D_1 \cdot m_1 + D_2 \cdot m_2 + \ldots$$

D_n : data input line

m_n : minterm formed by control input lines

Realizing Switching Functions with Multiplexers

Replacing the data input lines D_n with switching functions of p input variables leads arbitrary switching functions of p + q variables where q is the number of control inputs.

2 Sequential Circuits

2.1 Introduction and Definitions

- In **combinational** circuits, the output only depends on the **inputs at the same time**. This is **not the case** in sequential circuits.
- In sequential circuits, we have to introduce a **state** *Z* and we need to consider the time *t*.
- We assume a **central discrete clock,** i.e

  $t = 0,1,2,3,4,....$

2.1 (continued)

- The output depends on the state and on the input at the same time:

$$Y(t) = g(Z(t), X(t))$$

- The state at time *t* generally depends on **previous inputs** and on **previous states:**

$$Z(t+1) = f(Z(t), X(t))$$

- A systems that is governed my this equations are called **Mealy Machine**.

2.2 Mealy- and Moore Machine

- A special case is called **Moore Machine**.

  The output of a Moore Machine depends only on $Z(t)$:

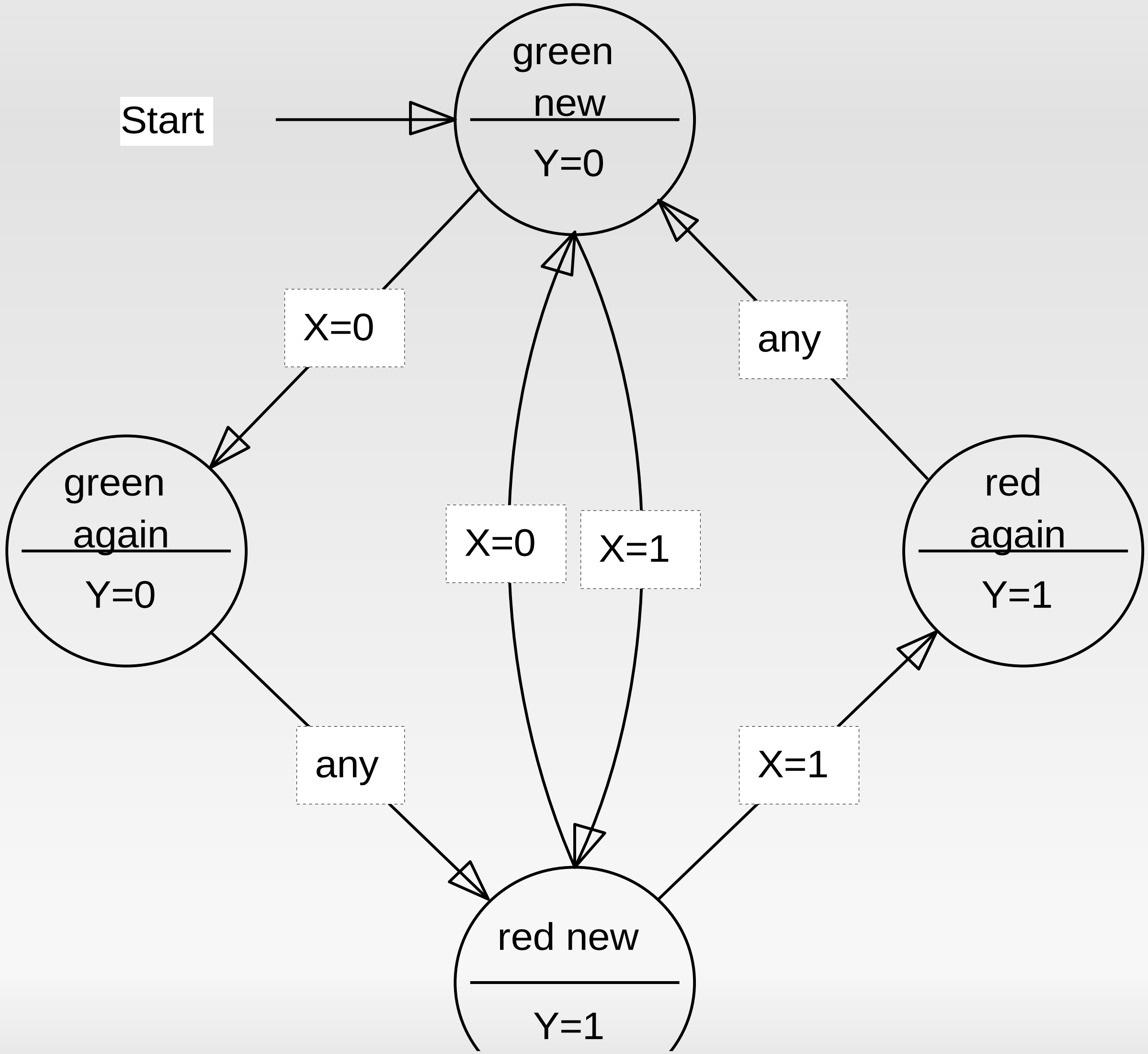
$$Z(t+1) = f(Z(t), X(t)) \qquad \text{(as before)}$$

$$Y(t) = g(Z(t)) \qquad \text{(depends only on } Z\text{)}$$

Moore Machine vs Mealy Machine

- The output of a Mealy Machine can change whenever the input changes or whenever a clock step occurs.
- The output output of a Moore Machine can **only change when a clock step** occurs. The output is **synchronized with the time grid**.
- In the state diagram of a **Mealy** Machine, the output values are associated with the **transition arcs**.
- In the state diagram of a Moore Machine, the output values are associated with the **state nodes**.

Example: Traffic Light



Circuit Level Design

Each state *Z* is represented by a combination of flip-flop output lines *Q0, Q1, Q2,...*

The problem of mapping of the flip-flop output values to the states is known as **Secondary State Assignment**. Different assignments lead to circuits of different complexity.

The rules for SSA are based on the concept of **adjacent codes**.

2.3 Secondary State Assignment

- Two present states should be assigned adjacent codes if they have the same next state for
  - a) Each input combination
  - b) Different input combinations, if the next state can also be given adjacent assignments
  - c) Some input combinations, but nonecessarily all
- For all inputs, codes assigned to the next states for each present state should be adjacent
- Assignments should simplify the output function.